

Face Recognition Vendor Test (FRVT)
2012

Still Face Image and Video
Concept, Evaluation Plan and API
Version 1.1

Patrick Grother, George W. Quinn, and Mei Ngan

Image Group
Information Access Division
Information Technology Laboratory



August 15, 2012

Status of this Document

This document is intended to be the final specification. August 15 revisions appear in blue. July 2012 changes appear in yellow. June 2012 changes are marked in cyan. Comments and questions should be submitted to frvt2012@nist.gov.

Timeline of the FRVT 2012 Evaluation

Phase 3	February to August 2013	NIST documentation and reports released.
	December 14	Open submission period ends – all implementations must be received by this date.
Phase 2 Oct 2012	October 31	Second interim report card released to submitting participants.
	October 15	Deadline for submission for inclusion of results in second interim report card.
Phase 1 July to Sept 2012	September 7	First interim report card released to submitting participants.
	August 28	Deadline for submission for inclusion of results in first interim report card.
Phase 0 April to June 2012	July 25	Open submission period begins.
	June 27	Final evaluation plan.
	June 8	Comments period closes on second draft of this document.
	May 15	Second draft evaluation plan (revised version of this document) for public comment.
	April 30	Request that participants give non-binding no-commitment indication of whether they will participate in the test.
		Comments period closes on first draft of this document.
	April 17	Initial draft evaluation plan (this document) for public comment.

July 2012							August 2012							September 2012							October 2012							November 2012							December 2012											
Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa					
1	2	3	4	5	6	7																																								
8	9	10	11	12	13	14		5	6	7	8	9	10	11		2	3	4	5	6	7	8						4	5	6	7	8	9	10												
15	16	17	18	19	20	21		12	13	14	15	16	17	18		9	10	11	12	13	14	15		14	15	16	17	18	19	20																
22	23	24	25	26	27	28		19	20	21	22	23	24	25		16	17	18	19	20	21	22		21	22	23	24	25	26	27																
29	30	31						26	27	28	29	30	31		23	24	25	26	27	28	29	30		28	29	30	31		25	26	27	28	29	30												
																30																														

Major Changes since MBE 2010

Please note that this document is derived from the MBE-STILL 2010 API document for continuity and to aid implementers of the FRVT 2012 API.

- For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux (see section 1.21).
- The FRVT 2012 API is written in the C++ language. Participants are required to provide their library in a format that is linkable using g++ (see 1.21).
- This evaluation contains new focus areas, which include:
 - Age, gender, and expression neutrality estimation for still images (see section 1.8)
 - Dedicated API for video data (see section 3.6)
 - Reporting minimum cost recognition (see section 1.16)
- New datasets will be used for FRVT 2012 and will contain individuals spanning a full age range.
- The header/source files for the API will be made available to implementers at <http://nigos.nist.gov:8080/frvt2012/>.

41 Table of Contents

42	1. FRVT.....	7
43	1.1. Scope.....	7
44	1.2. Audience	7
45	1.3. Market drivers	7
46	1.4. Offline testing	8
47	1.5. Phased testing.....	8
48	1.6. Interim reports.....	8
49	1.7. Final reports.....	8
50	1.8. Application scenarios.....	9
51	1.9. Image source labels	10
52	1.10. Options for participation	10
53	1.11. Number and schedule of submissions	10
54	1.12. Use of multiple images per person	11
55	1.13. Provision of photograph date information.....	11
56	1.14. Core accuracy metrics.....	11
57	1.15. Generalized accuracy metrics	12
58	1.16. Reporting minimum cost recognition	12
59	1.17. Reporting template size.....	13
60	1.18. Reporting computational efficiency.....	14
61	1.19. Exploring the accuracy-speed trade-space	14
62	1.20. Hardware specification	14
63	1.21. Operating system, compilation, and linking environment	14
64	1.22. Software and Documentation.....	15
65	1.23. Runtime behavior	16
66	1.24. Threaded computations.....	17
67	1.25. Time limits.....	17
68	1.26. Test datasets	18
69	1.27. Quality analysis	19
70	1.28. Ground truth integrity	19
71	2. Data structures supporting the API	19
72	2.1. Overview	19
73	2.2. Requirement	19
74	2.3. File formats and data structures.....	19
75	2.4. File structures for enrolled template collection	21
76	2.5. Data structure for result of an identification search	21
77	3. API Specification	22
78	3.1. Implementation identifiers	22
79	3.2. Maximum template size	22
80	3.3. 1:1 Verification.....	23
81	3.4. 1:N Identification	25
82	3.5. Pose conformance, age, gender, and expression neutrality estimation	31
83	3.6. Video.....	36
84	4. References.....	53
85	Annex A Submission of Implementations to the FRVT 2012	54
86	A.1 Submission of implementations to NIST	54
87	A.2 How to participate	54
88	A.3 Implementation validation	55
89		
90		
91		
92		
93		

94 List of Figures

95	Figure 1 – Organization and documentation of the FRVT 2012	7
96	Figure 2 – Notional DETs targeted by two different cost models	13
97	Figure 3 – Schematic of verification without enrollment database	23

99 List of Tables

100	Table 1 – Abbreviations	6
101	Table 2 – Subtests supported under the FRVT 2012 Still Image activity	9
102	Table 3 – FRVT 2012 classes of participation	10
103	Table 4 – Cumulative total number of algorithms, by class	10
104	Table 5 – Summary of accuracy metrics	12
105	Table 6 – Cost parameters for both submission types	13
106	Table 7 – Implementation library filename convention	15
107	Table 8 – Number of threads allowed for each application	17
108	Table 9 – Processing time limits in milliseconds	17
109	Table 10 – Main image corpora (others will be used)	18
110	Table 11 – Labels describing types of images	19
111	Table 12 – Structure for a single face	20
112	Table 13 – Structure for a set of images from a single person	20
113	Table 14 – Structure for a pair of eye coordinates	21
114	Table 15 – Enrollment dataset template manifest	21
115	Table 16 – Structure for a candidate	22
116	Table 17 – Implementation identifiers	22
117	Table 18 – Implementation template size requirements	22
118	Table 19 – Functional summary of the 1:1 application	23
119	Table 20 – SDK initialization	24
120	Table 21 – Template generation	24
121	Table 22 – Template matching	25
122	Table 23 – Procedural overview of the identification test	26
123	Table 24 – Enrollment initialization	27
124	Table 25 – Enrollment feature extraction	27
125	Table 26 – Enrollment finalization	28
126	Table 27 – Identification feature extraction initialization	29
127	Table 28 – Identification feature extraction	29
128	Table 29 – Identification initialization	30
129	Table 30 – Identification search	30
130	Table 31 – “Base” Estimator Class Structure	32
131	Table 32 – Example of SdkEstimator Class Declaration	33
132	Table 33 – Example of SdkEstimator Class Definition	33
133	Table 34 – Initialization of Pose conformance, Age, Gender, and Expression neutrality estimation	34
134	Table 35 – Pose conformance, Age, Gender, Expression neutrality estimation	34
135	Table 36 – API implementation requirements for Video	36
136	Table 37 – ONEVIDEO Class	37
137	Table 38 – EYEPAIR Class	38
138	Table 39 – PersonTrajectory typedef	38
139	Table 40 – PERSONREP Class	38
140	Table 41 – CANDIDATE Class	39
141	Table 42 – CANDIDATELIST typedef	39
142	Table 43 – ReturnCode class	39
143	Table 44 – VideoEnrollment::getPid	40
144	Table 45 – VideoEnrollment::initialize	41
145	Table 46 – VideoEnrollment::generateEnrollmentTemplate	41

146	Table 47 – VideoFinalize::finalize	42
147	Table 48 – VideoFeatureExtraction::initialize	43
148	Table 49 – VideoFeatureExtraction::generateIdTemplate	44
149	Table 50 – VideoSearch::initialize	45
150	Table 51 – VideoSearch::identifyVideo and VideoSearch::identifyImage	45
151	Table 52 – ImageEnrollment::getPid	46
152	Table 53 – ImageEnrollment::initialize	47
153	Table 54 – ImageEnrollment::generateEnrollmentTemplate.....	47
154	Table 55 – ImageFinalize::finalize.....	48
155	Table 56 – ImageFeatureExtraction::initialize	49
156	Table 57 – ImageFeatureExtraction::generateIdTemplate	50
157	Table 58 – ImageSearch::initialize	51
158	Table 59 – ImageSearch::identifyVideo.....	51
159		
160		

161 Acknowledgements

- 162 — The authors are grateful to the experts who made extensive comments on the first version of this document.

163 Project History

- 164 — July 31, 2012 – Release of additional information as API, v1.0
- 165 — June 27, 2012 – Release of revised API, v0.9.
- 166 — June 15, 2012 – Release of revised API, v0.7.
- 167 — May 15, 2012 – Release of revised API, v0.6.
- 168 — April 17, 2012 - Release of first public draft of the Face Recognition Vendor Test (FRVT) 2012 – Concept, Evaluation
- 169 Plan and API v0.5.
- 170 — June 17, 2010 – Published public report of MBE-STILL 2010 test (NISTIR 7709 – Report on the Evaluation of 2D Still-
- 171 Image Face Recognition Algorithms) linked from <http://face.nist.gov/mbe>.
- 172 — August 2009 - Briefed large scale 1:N proposal to U. S. Government sponsors

173 Terms and definitions

174 The abbreviations and acronyms of Table 1 are used in many parts of this document.

175 **Table 1 – Abbreviations**

FNIR	False negative identification rate
FPIR	False positive identification rate
FMR	False match rate
FNMR	False non-match rate
FRVT	NIST's Face Recognition Vendor Test program
FTS	Failure to Search
FTX	Failure to extract features from an enrollment image
GFAR	Generalized false accept rate
GFRR	Generalized false reject rate
DET	Detection error tradeoff characteristic: For verification this is a plot of FNMR vs. FMR (sometimes as normal deviates, sometimes on log-scales). For identification this is a plot of FNIR vs. FPIR.
INCITS	InterNational Committee on Information Technology Standards
ISO/IEC 19794	ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. (See Bibliography entry).
MBE	NIST's Multiple Biometric Evaluation program
NIST	National Institute of Standards and Technology
SDK	The term Software Development Kit refers to any library software submitted to NIST. This is used synonymously with the terms "implementation" and "implementation under test".

176

1. FRVT

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of face recognition implementations submitted to NIST's Face Recognition Vendor Test 2012. See

<http://www.nist.gov/itl/iad/ig/frvt-2012.cfm> for all FRVT 2012 documentation.

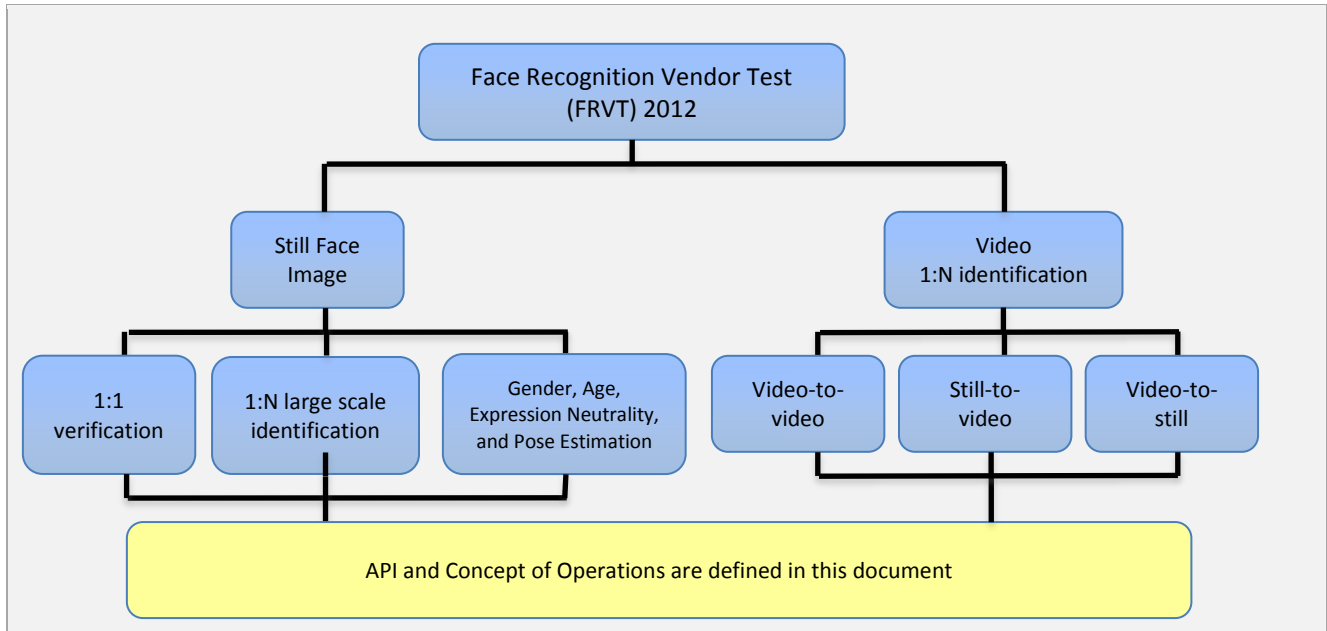


Figure 1 – Organization and documentation of the FRVT 2012

1.2. Audience

Universities and commercial entities with capabilities in any of the following areas are invited to participate in the FRVT 2012 Face test.

- Identity verification with face recognition algorithms.
- Large scale identification implementations.
- Profile view recognition.
- Those with a capability to assess age, gender, expression neutrality, and/or pose orientation of a face in an image.
- Face recognition in video capability

Organizations will need to implement the API defined in this document. Participation is open worldwide. There is no charge for participation. While NIST intends to evaluate technologies that could be readily made operational, the test is also open to experimental, prototype and other technologies.

1.3. Market drivers

This test is intended to support a plural marketplace of face recognition systems. While the dominant application, in terms of revenue, has been one-to-many search for driving licenses and visa issuance, the deployment of one-to-one face recognition has re-emerged with the advent of the e-Passport verification projects¹. In addition, there remains considerable activity in the use of FR for surveillance applications.

¹ These match images acquired from a person crossing a border against the ISO/IEC 19794-5 facial image stored on the embedded ISO/IEC 7816 + ISO/IEC ISO 14443 chips.

199 These applications are differentiated by the population size (and other variables). In the driving license duplicate
 200 detection application, the enrollment database might exceed 10^7 people. In the surveillance application, the watchlist
 201 size can readily extend to 10^4 .

202 **1.4. Offline testing**

203 While this set of tests is intended as much as possible to mimic operational reality, this remains an offline test executed
 204 on databases of images. The intent is to assess the core algorithmic capability of face recognition algorithms. This test will
 205 be conducted purely offline - it does not include a live human-presents-to-camera component. Offline testing is attractive
 206 because it allows uniform, fair, repeatable, and efficient evaluation of the underlying technologies. Testing of
 207 implementations under a fixed API allows for a detailed set of performance related parameters to be measured.

208 **1.5. Phased testing**

209 To support research and development efforts, this testing activity will embed multiple rounds of testing. These test
 210 rounds are intended to support improved performance. Once the test commences, NIST will evaluate implementations
 211 on a first-come-first-served basis and will return results to providers as expeditiously as possible. Providers may submit
 212 revised SDKs to NIST only after NIST provides results for the prior SDK and invites further submission. The frequency with
 213 which a provider may submit SDKs to NIST will depend on the times needed for developer preparation, transmission to
 214 NIST, validation, execution and scoring at NIST, and developer review and decision processes.

215 For the schedule and number of SDKs of each class that may be submitted, see sections 1.10 and 1.11.

216 **1.6. Interim reports**

217 The performance of each SDK will be reported in a "score-card". This will be provided to the participant. While the score
 218 cards may be used by the provider for arbitrary purposes, they are intended to facilitate development. Score cards will

- 219 – be machine generated (i.e. scripted),
- 220 – be provided to participants with identification of their implementation,
- 221 – include timing, accuracy and other performance results,
- 222 – include results from other implementations, but will not identify the other providers,
- 223 – be expanded and modified as revised implementations are tested, and as analyses are implemented,
- 224 – be generated and released asynchronously with SDK submissions,
- 225 – be produced independently of the other status of other providers' implementations,
- 226 – be regenerated on-the-fly, usually whenever any implementation completes testing, or when new analysis is added.

227 NIST does not intend to release these test reports publicly. NIST may release such information to the U.S. Government
 228 test sponsors. While these reports are not intended to be made public, NIST can only request that agencies not release
 229 this content.

230 **1.7. Final reports**

231 NIST will publish one or more final public reports. NIST may also

- 232 – publish additional supplementary reports (typically as numbered NIST Interagency Reports),
- 233 – publish in other academic journals,
- 234 – present results at conferences and workshops (typically PowerPoint).

235 Our intention is that the final test reports will publish results for the best-performing implementation from each
 236 participant. Because "best" is ill-defined (accuracy vs. time vs. template size, for example), the published reports may
 237 include results for other implementations. The intention is to report results for the most capable implementations (see
 238 section 1.14, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress
 239 or tradeoffs. **IMPORTANT:** Results will be attributed to the providers.

1.8. Application scenarios

The test will include one-to-one verification tests and one-to-many identification tests⁶ [MBE 2010, IREX III] for still images. It will also include one-to-many identification tests for video sequences. As described in Table 2, the test is intended to represent:

- Close-to-operational use of face recognition technologies in identification applications in which the enrolled dataset could contain images from up to three million persons.
- Verification scenarios in which still images are compared.
- Pose, age, gender, and expression neutrality estimation.
- Identification applications for face recognition in video

Table 2 – Subtests supported under the FRVT 2012 Still Image activity

#		A	B	C	D	V
1.	Aspect	1:1 verification	1:1 verification with enrollment database – Not Supported	1:N identification	Pose Conformance, Age, Gender, and Expression neutrality Estimation	Video-video, Still-video, video-still
2.	Enrollment dataset	None, application to single images	In MBE 2010, this class supported 1:1 verification with an enrollment database. This will not be supported for FRVT 2012.	N enrolled subjects	None, application to single images. Images will primarily be frontal controlled images (visa + mugshot) for which ground truth is known.	N enrolled sequences or N enrolled stills
3.	Prior NIST test references	Equivalent to 1 to 1 matching in [MBE 2010]		Equivalent to 1 to N matching in [MBE 2010]		
4.	Example application	Verification of e-Passport facial image against a live border-crossing image.		Open-set identification of an image against a central database, e.g. a search of a mugshot against a database of known criminals.	For sex and age: Digital signage for marketing. For pose and expression: Conformance to ISO/IEC 19794-5 requirements.	Open-set identification against a central database, e.g. a search of a wanted criminal through a live-video surveillance system at an airport who may attempt to flee the country
5.	Score or feature space normalization support	Vendor uses normalization techniques over SDK-internal datasets		Any score or feature based statistical normalization techniques are applied against enrollment database		Any score or feature based statistical normalization techniques are applied against enrollment database
6.	Intended number of subjects	Up to $O(10^5)$		Up to $O(10^7)$ but dependence on N will be computed. From $O(10^2)$ upwards.	Expected $O(10^3)$	Expected $O(10^3)$
7.	Number of images per individual	Variable, see section 1.12.		Variable, see section 1.12.	1	Variable

NOTE 1: The vast majority of images are color. The API supports both color and greyscale images.

NOTE 2: For the operational datasets, it is not known what processing was applied to the images before they were archived. So, for example, we do not know whether gamma correction was applied. NIST considers that best practice, standards and operational activity in the area of image preparation remains weak.

1.9. Image source labels

NIST may mix images from different source in an enrollment set. For example, NIST could combine N/2 mugshot images and N/2 visa images into a single enrollment dataset. For this reason, in the data structure defined in clause 2.3.3, each image is accompanied by a "label" which identifies the set-membership images. Legal values for labels are in clause 2.3.2.

1.10. Options for participation

The following rules apply:

- A participant must properly follow, complete and submit the Annex A Participation Agreement. This must be done once, not before July 18, 2012. It is not necessary to do this for each submitted SDK.
- All participants shall submit at least one class A SDK, or one class D SDK, or one class V SDK.
- A class A SDK shall be sent before, or concurrently with, any class C SDK.
- A class D SDK may be submitted without submission of a class A SDK.
- A class V SDK may be submitted without submission of a class A SDK.
- Any SDK shall implement exactly one of the functionalities defined in Table 3. So, for example, the 1:1 functionality of a class A SDK shall not be merged with that of a class C SDK.

Table 3 – FRVT 2012 classes of participation

Function	1:1 verification	1:1 verification with enrollment database	1:N identification	Pose conformance, Age, Gender, and Expression neutrality estimation	Video
Class label	A	B	C [CP & CN, see Table 6]	D	V
Co-requisite class SDK	None	Not Supported	A	None	None
API requirements	3.1 + 3.2 + 3.3	Not Supported	3.1 + 3.2 + 3.4	3.1 + 3.5	3.6

Class A might be preferred by academic institutions because the API is simple, supporting just the elemental hypothesis test: "are the images from the same person or not?"

1.11. Number and schedule of submissions

The test is conducted in three phases, as scheduled on page 2. The maximum total (i.e. cumulative) number of submissions is regulated in Table 4.

Table 4 – Cumulative total number of algorithms, by class

#	Phase 1	Total over Phases 1 + 2	Total over Phases 1 + 2 + 3
Cumulative total number of class A submissions	2	3	4 if at least 1 was successfully executed by end Phase 2 2 if zero had been successfully executed by end Phase 2
Cumulative total number of class C submissions	3 = 2CN + 1CP see sec. 1.16	4 = 3CN + 1CP (see sec 1.16)	7 = 5CN + 2CP if at least 1 was successfully executed by end Phase 2 3 = 2CN + 1CP if 0 had been successfully executed by end Phase 2
Cumulative total number of class D submissions	1	2	3
Cumulative total number of class V submissions	2	2	4 if at least 1 was successfully executed by end Phase 2 2 if zero had been successfully executed by end Phase 2

The numbers above may be increased as resources allow.

277 NIST cannot conduct surveys over runtime parameters - NIST must limit the extent to which participants are able to train
278 on the test data.

279 **1.12. Use of multiple images per person**

280 Some of the proposed datasets includes $K > 2$ images per person for some persons. This affords the possibility to model a
281 recognition scenario in which a new image of a person is compared against all prior images². Use of multiple images per
282 person has been shown to elevate accuracy over a single image [FRVT2002b, MBE 2010].

283 For **still-face recognition in** this test, NIST will enroll $K \geq 1$ images under each identity. Normally the probe will consist of a
284 single image, but NIST may examine the case that it could consist of multiple images. Ordinarily, the probe images will be
285 captured after the enrolled images of a person³. The method by which the face recognition implementation exploits
286 multiple images is not regulated: The test seeks to evaluate developer provided technology for multi-presentation fusion.
287 This departs from some prior NIST tests in which NIST executed fusion algorithms (e.g. [FRVT2002b]), and sum score
288 fusion, for example, [MINEX]).

289 This document defines a template to be the result of applying feature extraction to a set of $K \geq 1$ images. That is, a
290 template contains the features extracted from one or more images, not generally just one. An SDK might internally fuse K
291 feature sets into a single representation or maintain them separately - In any case the resulting proprietary template is
292 contained in a contiguous block of data. All verification and identification functions operate on such multi-image
293 templates.

294 The number of images per person will depend on the application area:

- 295 — In civil identity credentialing (e.g. passports, driving licenses) the images will be acquired approximately uniformly
296 over time (e.g. five years for a Canadian passport). While the distribution of dates for such images of a person might
297 be assumed uniform, a number of factors might undermine this assumption⁴.
- 298 — In criminal applications the number of images would depend on the number of arrests⁵. The distribution of dates for
299 arrest records for a person (i.e. the recidivism distribution) has been modeled using the exponential distribution, but
300 is recognized to be more complicated. NIST currently estimates that the number of images will never exceed 100.

301 **1.13. Provision of photograph date information to the implementation**

302 Due to face ageing effects, the utility of any particular enrollment image is dependent on the time elapsed between it and
303 the probe image. In FRVT 2012, NIST intends to use the most recent image as the probe image, and to use one or more of
304 the remaining prior images under a single enrolled identity.

305 **1.14. Core accuracy metrics**

306 Notionally the error rates for verification applications will be false match and false non-match error rates, FMR and FNMR.

307 For identification testing, the test will target open-universe applications such as benefits-fraud and watch-lists. It will not
308 address the closed-set task because it is operationally uncommon.

309 While some one-to-many applications operate with purely rank-based metrics, this test will primarily target score-based
310 identification metrics. Metrics are defined in Table 5. The analysis will survey over various rank and thresholds. Plots of
311 the two error rates, parametric on threshold, will be the primary reporting mechanism.

² For example, if a banned driver applies for a driving license under a new name, and the local driving license authority maintains a driving license system in which all previous driving license photographs are enrolled, then the fraudulent application might be detected if the new image matched any of the prior images. This example implies one (elemental) method of using the image history.

³ To mimic operational reality, NIST intends to maintain a causal relationship between probe and enrolled images. This means that the enrolled images of a person will be acquired before all the images that comprise a probe.

⁴ For example, a person might skip applying for a passport for one cycle (letting it expire). In addition, a person might submit identical images (from the same photography session) to consecutive passport applications at five year intervals.

⁵ A number of distributions have been considered to model recidivism, see "Random parameter stochastic process models of criminal careers." In Blumstein, Cohen, Roth & Visher (Eds.), Criminal Careers and Career Criminals, Washington, D.C.: National Academy of Sciences Press, 1986.

Table 5 – Summary of accuracy metrics

	Application	Metric	
A	1:1 Verification	FMR	= Fraction of impostor comparisons that produce a similarity score greater than or equal to a threshold value
		FNMR	= Fraction of genuine comparisons that produce a similarity score less than some threshold value
B	1:N Identification Primary identification metric	FPIR	= Fraction of searches that do not have an enrolled mate for which one or more candidate list entries is at or above a threshold
		FNIR	= Fraction of searches that have an enrolled mate for which the mate is below a threshold
C	1:N Identification (with rank criteria) Secondary identification metric	FPIR	= Fraction of searches that do not have an enrolled mate for which one or more candidate list entries is at or above a threshold
		FNIR	= Fraction of searches that have an enrolled mate for which the mate is not in the best R ranks <i>and</i> at or above a threshold

NOTE: The metric on line B is a special case of the metric on line C: the rank condition is relaxed ($R \rightarrow N$). Metric B is the primary metric of interest because the target application does not include a rank criterion.

FPIR will be estimated using probe images for which there is no enrolled mate.

NIST will extend the analysis in other areas, with other metrics, and in response to the experimental data and results.

1.15. Generalized accuracy metrics

Under the ISO/IEC 19795-1 biometric testing and reporting standard, a test must account for "failure to acquire" (FTA) and "failure to enroll" (FTE) events (e.g. elective refusal to make a template, or fatal errors). The way these are treated is application-dependent.

For verification, the appropriate metrics reported in FRVT 2012 will be generalized error rates (GFAR, GFRR). When single images are compared, (GFAR,GFRR) and (FMR,FNMR) will be equivalent if no failures are observed.

Similarly for identification, generalized error rates will be reported.

1.16. Reporting minimum cost recognition

This evaluation will investigate the use of cost parameters for application-specific algorithm optimization. The goal is to determine if matching algorithms can be modified to improve performance when the costs of errors are known in advance. The following cost model will be used as an evaluation metric for recognition performance:

$$E[\text{Cost}(\tau)] = (1 - P_{\text{Mated}}) \text{FPIR}(\tau) C_P + P_{\text{Mated}} \text{FNIR}(\tau) C_N$$

where P_{Mated} is the *a priori* probability that the user is mated, C_P is the cost of a false positive, C_N is the cost of a false negative, $\text{FPIR}(\tau)$ is the false positive identification rate, $\text{FNIR}(\tau)$ is the false negative identification rate, and τ is the operating threshold. The model estimates the expected cost per user attempt, which could be a measure of time, workload, money, etc. The participant is tasked with minimizing the cost for a predetermined and fixed set of cost parameters (C_P , C_N , and P_{Mated}).

Cost parameters are often chosen to correspond to a specific application. Consider a biometric system that provides bank vault access to specific individuals. One might reasonably set the cost of a false positive to be the monetary value of whatever is in the vault, and the cost of a false negative to a value that reflects the amount of inconvenience incurred from having to open the vault by some other method. Setting P_{Mated} to 0.1 assumes that one out of every ten access attempts is by an allowed user.

NIST **recommends requires** each participant to submit instances of the class C SDK, each corresponding to a different set of cost parameters. These parameters are defined in the table below. Class CP implementations penalize false positives heavily and false negatives lightly. Class CN implementations assign comparatively greater penalty to false negatives. For this class of implementations, suppression of false positives is less important.

Table 6 – Cost parameters for both submission types

Implementation Class	CN	CP	P _{Mated}
Class CP	1	1000	0.6
Class CN	250	1	0.001

Additionally, failures to extract (FTXs) and failures to search (FTSs) will be treated differently depending on the implementation class.

- For Class CP implementations, both will be treated as failures in a positive recognition system (e.g. access control). This is the way NIST has handled FTXs and FTSs in prior evaluations.
- For Class CN implementations, FTXs and FTSs will be treated like failures in a negative recognition system (e.g. a watchlist). Failures in a negative recognition system increase the FPIR when they occur for non-mated searches, but do not increase the FNIR when they occur for mated searches. This differs from the way NIST has traditionally handled these types of failure.

The motivation for requiring participants to submit two implementations is to see if it is possible to change the shape of a DET to reduce cost for a specific set of cost parameters. Figure 2 plots standard DET curves for two identification algorithms. The two curves cross one another, making it impossible to state which is more accurate in any absolute sense. Since class CN implementations are penalized heavily for false negatives, and only lightly for false positives, both algorithms are expected to achieve their lowest cost toward the right end of the figure, where the blue curve performs better. Conversely, class CP implementations are penalized heavily for false positives but only lightly for false negatives. Thus, for this set of cost parameters, both algorithms are expected to achieve their lowest cost toward the left end of the figure, where the red curve performs better.

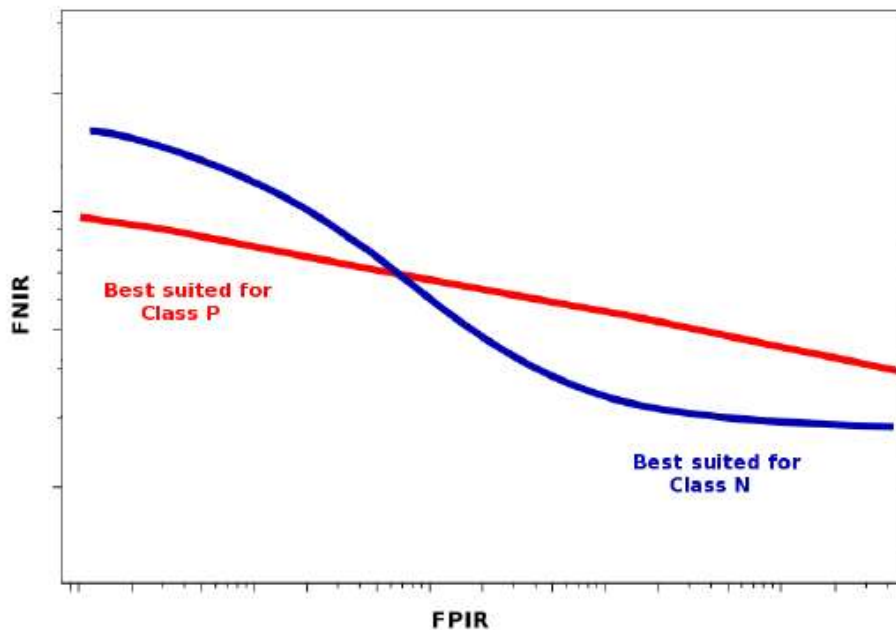


Figure 2 – Notional DETs targeted by two different cost models

1.17. Reporting template size

Because template size is influential on storage requirements and computational efficiency, this API supports measurement of template size. NIST will report statistics on the actual sizes of templates produced by face recognition

implementations submitted to FRVT 2012. NIST may report statistics on runtime memory usage. Template sizes were reported in the IREX III test⁶ and the MBE-STILL 2010 test⁷.

1.18. Reporting computational efficiency

As with other tests, NIST will compute and report recognition accuracy. In addition, NIST will also report timing statistics for all core functions of the submitted SDK implementations. This includes feature extraction, 1:1 and 1:N recognition, and age, gender, pose frontality and expression neutrality estimation. For an example of how efficiency can be reported, see the final report of the IREX III test⁶ and the MBE-STILL 2010 test⁷.

Note that face recognition applications optimized for pipelined 1:N searches may not demonstrate their efficiency in pure 1:1 comparison applications.

1.19. Exploring the accuracy-speed trade-space

NIST will explore the accuracy vs. speed tradeoff for face recognition algorithms running on a fixed platform. NIST will report both accuracy and speed of the implementations tested. While NIST cannot force submission of "fast vs. slow" variants, participants may choose to submit variants on some other axis (e.g. "experimental vs. mature") implementations. NIST encourages "fast-less-accurate vs. slow-more-accurate" with a factor of three between the speed of the fast and slow versions.

1.20. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The blades are labeled as Dell M905, M910, M605, and M610. The following list gives some details about the hardware of each blade type:

- Dell M605 - Dual Intel Xeon E5405 2 GHz CPUs (4 cores each)
- Dell M610 - Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each)
- Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs⁸ (4 cores each)
- Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)

Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. We anticipate that 16 processes can be run without time slicing.

NIST is requiring use of 64 bit implementations throughout. This will support large memory allocation to support 1:N identification task with image counts in the millions. For still images, if all templates were to be held in memory, the 192GB capacity implies a limit of ~19KB per template, for a 10 million image enrollment. For video, given the data expectations and the occurrence of faces in the imagery, we anticipate the developers will have sufficient memory for video templates. Note that while the API allows read access of the disk during the 1:N search, the disk is, of course, relatively slow.

Some of the section 3 API calls allow the implementation to write persistent data to hard disk. The amount of data shall not exceed 200 kilobytes per enrolled image. NIST will respond to prospective participants' questions on the hardware, by amending this section.

1.21. Operating system, compilation, and linking environment

The operating system that the submitted implementations shall run on will be released as a downloadable file accessible from <http://nigos.nist.gov:8080/evaluations/> which is the 64-bit version of CentOS 6.2 running Linux kernel 2.6.32-220.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux.

⁶ See the IREX III test report: NIST Interagency Report 7836, linked from <http://iris.nist.gov/irex>

⁷ See the MBE-STILL 2010 test report, NIST Interagency Report 7709, linked from <http://face.nist.gov/mbe>

⁸ `cat /proc/cpuinfo` returns `fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3wext 3dnow constant_tsc nonstop_tsc pni cx16 popcnt lahf_lm cmp_legacy svm extapic cr8_legacy altmovcr8 abm sse4a misalignsse 3dnowprefetch osvw`

410 NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library
 411 in a format that is linkable using g++ version 4.4.6. The standard libraries are:

412 `/usr/lib64/libstdc++.so.6.0.13 lib64/libc.so.6 -> libc-2.12.so lib64/libm.so.6 -> libm-2.12.so`

413 A typical link line might be

414 `g++ -l. -Wall -m64 -o frvt12test frvt12test.cpp -L. -lfrvt2012_Enron_A_07`

415 The Standard C++ library should be used for development of the SDKs. The prototypes from the still image API portion of
 416 this document will be written to a file "frvt2012.h" which will be included via

```
#include <frvt2012.h>
```

417 The prototypes from the video API portion of this document will be written to a file "frvt2012Video.h" which will be
 418 included via

```
#include <frvt2012Video.h>
```

419 The header files will be made available to implementers at <http://nigos.nist.gov:8080/frvt2012/>.

420 NIST will handle all input of images via the JPEG and PNG libraries, sourced, respectively from <http://www.iij.org/> and see
 421 <http://libpng.org>.

422 All compilation and testing will be performed on x86 platforms. Thus, participants are strongly advised to verify library-
 423 level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to avoid linkage
 424 problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

425 Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are
 426 discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

427 1.22. Software and Documentation

428 1.22.1. SDK Library and Platform Requirements

429 Participants shall provide NIST with binary code only (i.e. no source code). Header files (".h") are allowed, but these shall
 430 not contain intellectual property of the company nor any material that is otherwise proprietary. It is preferred that the
 431 SDK be submitted in the form of a single static library file. However, dynamically linked shared library files are permitted.

432 The core library shall be named according to Table 7. Additional shared object library files may be submitted that support
 433 this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

434 Intel Integrated Performance Primitives (IPP) libraries are permitted if they are delivered as a part of the developer-
 435 supplied library package. It is the provider's responsibility to establish proper licensing of all libraries. The use of IPP
 436 libraries shall not inhibit the SDK's ability to run on CPUs that do not support IPP. Please take note that some IPP
 437 functions are multithreaded and threaded implementations may complicate comparative timing.

438 Access to any GPUs is not permitted.

439 **Table 7 – Implementation library filename convention**

Form	libFRVT2012_provider_class_sequence.ending				
Underscore delimited parts of the filename	libFRVT2012	provider	class	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Acme	Function classes supported in Table 3. EXAMPLE: C	A two digit decimal identifier to start at 00 and increment by 1 every time any SDK is sent to NIST. EXAMPLE: 07	Either .so or .a
Example	libFRVT2012_Acme_C_07.a				

440

441 NIST will report the size of the supplied libraries.

442 **1.22.2. Configuration and developer-defined data**

443 The implementation under test may be supplied with configuration files and supporting data files. The total size of the
444 SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes =
445 1024^3 bytes.

446 NIST will report the size of the supplied configuration files.

447 **1.22.3. Installation and Usage**

448 The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be
449 executable on any number of machines without requiring additional machine-specific license control procedures or
450 activation.

451 The SDK shall be installable using simple file copy methods. It shall not require the use of a separate installation program.

452 The SDK shall neither implement nor enforce any usage controls or limits based on licenses, number of executions,
453 presence of temporary files, etc. The SDKs shall remain operable until April 30 2013.

454 Hardware (e.g. USB) activation dongles are not acceptable.

455 **1.22.4. Hard disk space**

456 FRVT 2012 participants should inform NIST if their implementations require more than 100K of persistent storage, per
457 enrolled image on average.

458 **1.22.5. Documentation**

459 Participants shall provide complete documentation of the SDK and detail any additional functionality or behavior beyond
460 that specified here. The documentation must define all (non-zero) developer-defined error or warning return codes.

461 **1.22.6. Modes of operation**

462 Individual SDKs provided shall not include multiple “modes” of operation, or algorithm variations. No switches or options
463 will be tolerated within one library. For example, the use of two different “coders” by a feature extractor must be split
464 across two separate SDK libraries, and two separate submissions.

465 **1.22.7. Watermarking of images**

466 The SDK functions shall not watermark or otherwise steganographically mark up the images.

467 **1.23. Runtime behavior**

468 **1.23.1. Interactive behavior**

469 The SDK will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall
470 not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal
471 interaction e.g. reads from “standard input”.

472 **1.23.2. Error codes and status messages**

473 The SDK will be tested in non-interactive “batch” mode, without terminal support. Thus, the submitted library shall run
474 quietly, i.e. it should not write messages to “standard error” and shall not write to “standard output”. An SDK may write
475 debugging messages to a log file - the name of the file must be declared in documentation.

476 **1.23.3. Exception Handling**

477 The application should include error/exception handling so that in the case of a fatal error, the return code is still
478 provided to the calling application.

1.23.4. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process), nor read from such. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.23.5. Stateless behavior

All components in this test shall be stateless, except as noted. This applies to face detection, feature extraction and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.24. Threaded computations

Table 8 shows the limits on the numbers of threads a face recognition implementation may use. In many cases threading is not permitted (i.e. $T=1$) because NIST will parallelize the test by dividing the workload across many cores and many machines. For the functions where we allow multi-threading, e.g. in the 1:N test, NIST requires the provider to disclose the maximum number of threads to us. If that number is T , NIST will run the largest integer number of processes, P , in parallel such that $TP \leq 16$.

Table 8 – Number of threads allowed for each application

	A	C	D	V
Function	1:1 verification	1:N identification	Pose conformance, Age, Gender, Expression neutrality estimation	Video
Feature extraction	1	1	1	1
Verification	1	NA		NA
Finalize enrollment (before 1:1 or 1:N)	NA	$1 \leq T \leq 16$		$1 \leq T \leq 16$
Identification	NA	$1 \leq T \leq 16$		$1 \leq T \leq 16$

498

For comparative timing, the IREX III⁶ test report estimated a factor by which the speed of threaded algorithms would be adjusted. Non-threaded implementations will eliminate the need for NIST to apply such techniques [IREX III].

NIST will not run an implementation from participant X and an implementation from participant Y on the same machine at the same time.

To expedite testing, for single-threaded libraries, NIST will run up to $P = 16$ processes concurrently. NIST's calling applications are single-threaded.

1.25. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 9. These time limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are 90-th percentiles. This means that 90% of all operations should take less than the identified duration.

The time limits apply per image. When K images of a person are present, the time limits shall be increased by a factor K .

Table 9 – Processing time limits in milliseconds

	A	C	D	V
Function	1:1 verification	1:N identification	Pose	Video

511

	without enrollment database		conformance, Age, Gender, and Expression neutrality estimation	
Feature extraction enrollment	1000 (1 core)	1000 (1 core)	500 (1 core)	5 * class C per video frame
Feature extraction for verification or identification	1000 (1 core)	1000 (1 core)		5 * class C per video frame
Verification	5 (1 core)	NA		NA
Identification of one search image against 1,000,000 single-image MULTIFACE records.	NA	10000 (16 cores) or 160000 (1 core)		NA

512 For video: the multiple of 5 is a notional average of the number of persons expected in any given frame. This figure is
513 highly unreliable for any given sample.

514 In addition the enrollment finalization procedure is subject to a time limit, as follows. For an enrollment of one million
515 single-image **MULTIFACE**s, the total time shall be less than 7200 seconds. The implementation can use up to 16 cores.
516 This limit includes disk IO time.

517 1.26. Test datasets

518 This section is under development. The data has, in some cases, been estimated from initial small partitions. The
519 completion of this section depends on further work. The information is subject to change. We intend to update this
520 section as fully as possible.

521 NIST is likely to use other datasets, in addition. Information for video data is given in section 3.6.

522 **Table 10 – Main image corpora (others will be used)**

	Laboratory	FRVT 2002+2006 / HCINT	New Dataset	Multiple Encounter Database (MEDS)
Collection, environment	See FRVT 2006 Report, Phillips et al. NIST IR 7408.	Visa application process	Visa application process	Law enforcement booking
Live scan, Paper		Live	Live	Live, few paper
Documentation		See NIST IR 6965 [FRVT2002]	New	See NIST Special Database 32 Volume 1 (MEDS-I) and Volume 2 (MEDS-II) ⁹ .
Compression from [MBE 2010] ¹⁰		JPEG mean size 9467 bytes. See [FRVT2002b]	JPEG mean size 17 kilobytes	JPEG ~ 20:1
Maximum image size		300 x 252	300 x 252	Mixed, some are 640x480 others are 768x960, some are smaller.
Minimum image size		300 x 252	300 x 252	
Eye to eye distance		Median = 71 pixels	Median = 71 pixels	mean=156, sd=46
Frontal		Yes, well controlled		Moderately well controlled Profile images will be included and labeled as such.
Full frontal geometry		Yes, in most cases. Faces may have small background than ISO FF requires.	Yes, in most cases. Faces may have small background than ISO FF requires.	Mostly not. Varying amounts of the torso are visible.
Intended use	1:1	1:1 and 1:N		1:N

⁹ NIST Special Database 32, Volume 1 and Volume 2 are available at: <http://www.nist.gov/itl/iad/ig/sd32.cfm>. MEDS-II is an update to MEDS-I and was published in February 2011. Note that NIST does not provide "training" data per se - this differs from the paradigm often used in academic research where a model is trained, tested and validated. Instead FRVT 2012 follows operational reality: software is typically shipped "as is" with a fixed internal representation that is designed to be usable "off the shelf" without training and with only minimal configuration.

¹⁰ Compression effects were studied under MBE 2010 in NIST Interagency Report 7830, linked from <http://face.nist.gov/mbc>

Age	University population	18 years and above	0 years and above	18 years and above
-----	-----------------------	--------------------	-------------------	--------------------

523 1.27. Quality analysis

524 NIST will examine the effectiveness of quality scores in predicting recognition accuracy. A quality score is computed from
 525 an input record during feature extraction. The default method of analysis will be the error vs. reject analysis document in
 526 P. Grother and E. Tabassi, *Performance of biometric quality measures*, IEEE Trans. PAMI, 29:531–543, 2007.

527 The default use-case is that the enrollment image is assumed to be pristine (in conformance with the ISO standard, for
 528 example), and quality is being used *during* a verification or identification transaction to select the image most likely to
 529 match the reference image. The reference image is assumed to be unavailable for matching during the collection.

530 For reasons of operational realism, metadata, such as a date of birth, will not be provided to the quality computation.

531 Analyses other than for the default case may be conducted.

532 1.28. Ground truth integrity

533 Some of the test databases will be derived from operational systems. They may contain ground truth errors in which

- 534 — a single person is present under two different identifiers, or
- 535 — two persons are present under one identifier, or
- 536 — in which a face is not present in the image.

537 If these errors are detected, they will be removed. NIST will use aberrant scores (high impostor scores, low genuine
 538 scores) to detect such errors. This process will be imperfect, and residual errors are likely. For comparative testing,
 539 identical datasets will be used and the presence of errors should give an additive increment to all error rates. For very
 540 accurate implementations this will dominate the error rate. NIST intends to attach appropriate caveats to the accuracy
 541 results. For prediction of operational performance, the presence of errors gives incorrect estimates of performance.

542 2. Data structures supporting the API

543 2.1. Overview

544 This section describes separate APIs for the core face recognition applications described in section 1.8. All SDK's
 545 submitted to FRVT 2012 shall implement the functions required by the rules for participation listed before Table 3.

546 2.2. Requirement

547 FRVT 2012 participants shall submit an SDK which implements the relevant C++ prototyped interfaces of clause 3. C++
 548 was chosen in order to make use of some object-oriented features.

549 2.3. File formats and data structures

550 2.3.1. Overview

551 In this face recognition test, an individual is represented by $K \geq 1$ two-dimensional facial images, and by subject and
 552 image-specific metadata.

553 2.3.2. Dictionary of terms describing images

554 Images will be accompanied by one of the labels given in Table 11. Face recognition implementations submitted to FRVT
 555 2012 should tolerate images of any category.

556 **Table 11 – Labels describing types of images**

	Label as C++ string	Primary test area	Meaning
1.	"unknown"		Either the label is unknown or unassigned.

2.	"laboratory frontal controlled"	1:1	Frontal with controlled illumination
3.	"laboratory frontal uncontrolled"	1:1	Any illumination
4.	"laboratory nonfrontal controlled"	1:1	NOTE: There is no hyphen "-"
5.	"laboratory nonfrontal uncontrolled"	1:1	Any illumination, pose is unknown and could be frontal
6.	"visa"	1:N	Either a member of the FRVT 2002/2006 HCINT corpus or one of similar properties.
7.	"mugshot"	1:N	Either a member of the Multi-encounter law enforcement database or one of similar properties. The image is nominally frontal - See NIST Special Database 32 ⁹ .
8.	"profile"	1:N	The image is a profile image taken from the multi-encounter law enforcement database.

557

558 NIST intends to use "profile" images in this evaluation.

559 **2.3.3. Data structures for encapsulating multiple images**

560 The standardized formats for facial images are the ISO/IEC 19794-5:2005 and the ANSI/NIST ITL 1-2007 type 10 record.

561 The ISO record can store multiple images of an individual in a standalone binary file. In the ANSI/NIST realm, K images of
562 an individual are usually represented as the concatenation of one Type 1 record + K Type 10 records. The result is usually
563 stored as an EFT file.564 An alternative method of representing K images of an individual is to define a structure containing an image filename and
565 metadata fields. Each file contains a standardized image format, e.g. PNG (lossless) or JPEG (lossy).

566

Table 12 – Structure for a single face

567

Removed fields: dob, mob, yob, day, month, year, sex, race, height, and weight

	C++ code fragment	Remarks
1.	typedef struct sface	
2.	{	
3.	uint16_t image_width;	Number of pixels horizontally
4.	uint16_t image_height;	Number of pixels vertically
5.	uint16_t image_depth;	Number of bits per pixel. Legal values are 8 and 24.
6.	uint8_t format;	Flag indicating native format of the image as supplied to NIST 0x01 = JPEG (i.e. compressed data) 0x02 = PNG (i.e. never compressed data)
7.	uint8_t *data;	Pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes I I I I I I I I
8.	string description;	Single description of the image. The allowed values for this string are given in Table 11.
9.		
10.	} ONEFACE;	

568

Table 13 – Structure for a set of images from a single person

569

Removed fields: numfaces

570

Please note the change from struct [MBE 2010] to typedef [FRVT 2012] for this data structure.

	C++ code fragment	Remarks
1.	typedef std::vector<ONEFACE> MULTIFACE;	Vector containing F pre-allocated face images of the same person. The number of items stored in the vector is accessible via the vector::size() function.

571 **2.3.4. Data structure for eye coordinates**572 SDKs should return eye coordinates of each enrolled facial image. This function, while not necessary for a recognition
573 test, will assist NIST in assuring the correctness of the test database. The primary mode of use will be for NIST to inspect
574 images for which eye coordinates are not returned, or differ between developer SDKs.

575 The eye coordinates shall follow the placement semantics of the ISO/IEC 19794-5:2005 standard - the geometric
 576 midpoints of the endocanthion and exocanthion (see clause 5.6.4 of the ISO standard).

577 Sense: The label "left" refers to subject's left eye (and similarly for the right eye), such that $x_{right} < x_{left}$.

578 **Table 14 – Structure for a pair of eye coordinates**

	C++ code fragment	Remarks
1.	typedef struct ohos	
2.	{	
	bool failed;	If the eye coordinates have been computed and assigned successfully, this value should be set to false, otherwise true.
3.	int16_t xleft;	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. $x < 0$ or $x \geq \text{width}$) indicate the implementation believes the eye center is outside the image.
4.	int16_t yleft;	
5.	int16_t xright;	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. $x < 0$ or $x \geq \text{width}$) indicate the implementation believes the eye center is outside the image.
6.	int16_t yright;	
7.	} EYEPAIR;	

579 2.3.5. Data type for similarity scores

580 Identification and verification functions shall return a measure of the similarity between the face data contained in the
 581 two templates. The datatype shall be an eight byte double precision real. The legal range is [0, DBL_MAX], where the
 582 DBL_MAX constant is larger than practically needed and defined in the <limits.h> include file. Larger values indicate more
 583 likelihood that the two samples are from the same person.

584 Providers are cautioned that algorithms that natively produce few unique values (e.g. integers on [0,127]) will be
 585 disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly
 586 0.0001, for example.

587 2.4. File structures for enrolled template collection

588 An SDK converts a **MULTIFACE** into a template, using, for example the "convert_**MULTIFACE**_to_enrollment_template"
 589 function of section 3.4.3. To support the class C identification functions of Table 3, NIST will concatenate enrollment
 590 templates into a single large file. This file is called the EDB (for enrollment database). The EDB is a simple binary
 591 concatenation of proprietary templates. There is no header. There are no delimiters. The EDB may extend to hundreds of
 592 gigabytes in length

593 This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest
 594 has the format shown as an example in Table 15. If the EDB contains N templates, the manifest will contain N lines. The
 595 fields are space (ASCII decimal 32) delimited. There are three fields, all containing numeric integers. Strictly speaking, the
 596 third column is redundant.

597 **Table 15 – Enrollment dataset template manifest**

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	Unsigned decimal integer	Unsigned decimal integer	Unsigned decimal integer
Datatype length required	4 bytes	4 bytes	8 bytes
Example lines of a manifest file appear to the right. Lines 1, 2, 3 and N appear.	90201744	1024	0
	163232021	1536	1024
	7456433	512	2560
	...		
	183838	1024	30720000

598
 599 The EDB scheme avoids the file system overhead associated with storing millions of individual files.

600 2.5. Data structure for result of an identification search

601 All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most
 602 similar matching entries list first with lowest rank. The data structure shall be that of Table 16.

Table 16 – Structure for a candidate

	C++ code fragment	Remarks
1.	typedef struct candidate	
2.	{	
3.	bool failed;	If the candidate computation failed, this value is set to true. If the candidate is valid it should be set to false.
4.	uint32_t template_id;	The Template ID integer from the enrollment database manifest defined in clause 0.
5.	double similarity_score;	Measure of similarity between the identification template and the enrolled candidate. Higher scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	double probability;	An estimate of the probability that the biometric data and candidate belong to different persons, i.e. the probability that a score this large would be observed given that the pair of images are from different people = $P(\text{SCORE} \mid \text{IMPOSTOR})$. This value shall be on [0:1]. This is one minus the integral of the expected impostor distribution from 0 to the similarity score, i.e. the expected false match rate.
7.	} CANDIDATE;	

3. API Specification

3.1. Implementation identifiers

All implementations shall support the self-identification function of Table 17. This function is required to support internal NIST book-keeping. The version numbers should be distinct between any versions, which offer different algorithmic functionality.

Table 17 – Implementation identifiers

Prototype	int32_t get_pid(string &sdk_identifier, string &email_address);	
		A developer-assigned ID. This shall be different for each submitted SDK.
		Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdk_identifier	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email_address	Point of contact email address. The value cannot be the empty string.
Return Value	0	Success
	Other	Vendor-defined failure

3.2. Maximum template size

All implementations shall report the maximum expected template sizes. These values will be used by the NIST test harnesses to pre-allocate template data. The values should apply to a single image. For a **MULTIFACE** containing K images, NIST will allocate K times the value returned. The function call is given in Table 18.

Table 18 – Implementation template size requirements

Prototype	int32_t get_max_template_sizes(uint32_t &max_enrollment_template_size, uint32_t &max_recognition_template_size)	
		Output
		Output
Description	This function retrieves the maximum template size needed by the feature extraction routines.	
Output	max_enrollment_template_size	The maximum possible size, in bytes, of the memory needed to store feature

Parameters		data from a single enrollment image.
	max_recognition_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single verification or identification image.
Return Value	0	Success
	Other	Vendor-defined failure

3.3. 1:1 Verification

3.3.1. Overview

The 1:1 testing will proceed in three phases: preparation of enrollment templates; preparation of verification templates; and matching. These are detailed in Table 19.

Table 19 – Functional summary of the 1:1 application

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Initialization	I1	Initialization	Function to allow implementation to read configuration data, if any.	None
Enrollment	E1	Serial enrollment	Given $K \geq 1$ input images of an individual, the implementation will create a proprietary enrollment template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Verification	V1	Serial verification	Given $K \geq 1$ input images of an individual, the implementation will create a proprietary verification template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Matching (i.e. comparison)	C1	Serial matching	Given one proprietary enrollment template and one proprietary verification template, compare these and produce a similarity score. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs.

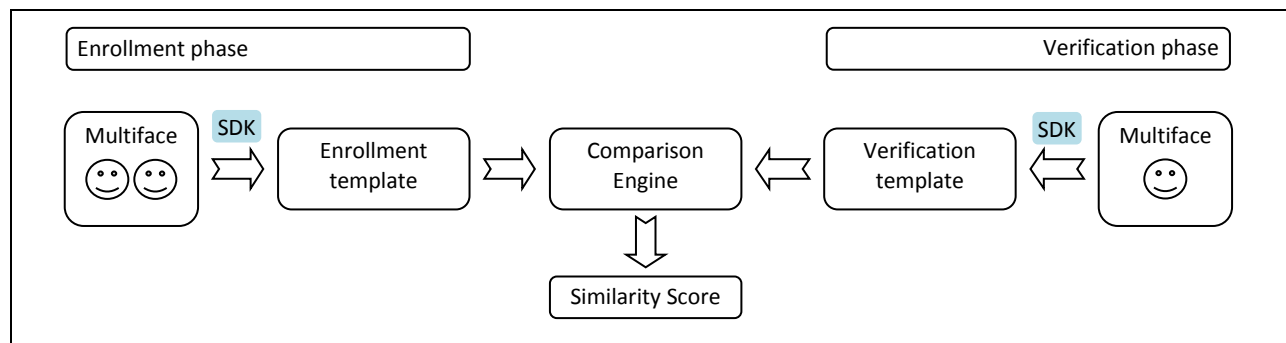


Figure 3 – Schematic of verification without enrollment database

3.3.2. API

Initialization of the implementation

Before any template generation or matching calls are made, the NIST test harness will make a call to the initialization of the function in Table 20.

Table 20 – SDK initialization

Removed fields: num_descriptions

Prototype	int32_t initialize_verification(const string &configuration_location, const std::vector<string> &descriptions);	
		Input
		Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the Table 21 functions convert MULTIFACE _to_enrollment_template or convert MULTIFACE _to_verification_template. The SDK under test should set all parameters.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
	descriptions	A lexicon of labels one of which will be assigned to each image. EXAMPLE: The descriptions could be {"mugshot", "visa", "unknown"}. These labels are provided to the SDK so that it knows to expect images of these kinds. The number of items stored in the vector is accessible via the vector::size() function.
Output Parameters	none	
Return Value	0	Success
	2	Vendor provided configuration files are not readable in the indicated location.
	8	The descriptions are unexpected, or unusable.
	Other	Vendor-defined failure

Template generation

The functions of Table 21 support role-specific generation of a template data. The format of the templates is entirely proprietary.

Table 21 – Template generation

Prototypes	int32_t convert_ MULTIFACE _to_enrollment_template(const MULTIFACE &input_faces, uint32_t &template_size, uint8_t *proprietary_template);	
		Input
		Output
		Output
	int32_t convert_ MULTIFACE _to_verification_template(const MULTIFACE &input_faces, uint32_t &template_size, uint8_t *proprietary_template, uint8_t &quality);	
		Output
Description	This function takes a MULTIFACE , and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. In all cases, even when unable to extract features, the output shall be a template record that may be passed to the match_templates function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this.	
Input Parameters	input_faces	An instance of a Table 13 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	template_size	The size, in bytes, of the output template
	proprietary_template	The output template. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the MULTIFACE ; the value T is output by the maximum template size functions of Table 18.
	quality	An assessment of image quality. This is optional. The legal values are – [0,100] - The value should have a monotonic decreasing relationship with false non-match rate anticipated for this sample if it was compared with a pristine image of the same

		person. So, a low value indicates high expected FNMR. – 255 - This value indicates a failed attempt to calculate a quality score. – 254 - This values indicates the value was not assigned.
Return Value	0	Success
	2	Elective refusal to process this kind of MULTIFACE
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

634

635 Matching

636 Matching of one enrollment against one verification template shall be implemented by the function of Table 22.

637

Table 22 – Template matching

Prototype	int32_t match_templates(const uint8_t *verification_template, const uint32_t verification_template_size, const uint8_t *enrollment_template, const uint32_t enrollment_template_size, double &similarity);	
		Input
		Input
		Input
		Input
		Output
Description	This function compares two opaque proprietary templates and outputs a similarity score, which need not satisfy the metric properties. NIST will allocate memory for this parameter before the call. When either or both of the input templates are the result of a failed template generation (see Table 21), the similarity score shall be -1 and the function return value shall be 2.	
Input Parameters	verification_template	A template from convert MULTIFACE _to_verification_template().
	verification_template_size	The size, in bytes, of the input verification template $0 \leq N \leq 2^{32} - 1$
	enrollment_template	A template from convert MULTIFACE _to_enrollment_template().
	enrollment_template_size	The size, in bytes, of the input enrollment template $0 \leq N \leq 2^{32} - 1$
Output Parameters	similarity	A similarity score resulting from comparison of the templates, on the range [0,DBL_MAX]. See section 2.3.5.
Return Value	0	Success
	2	Either or both of the input templates were result of failed feature extraction
	Other	Vendor-defined failure

638 **3.4. 1:N Identification**639 **3.4.1. Overview**640 The 1:N application proceeds in two phases, enrollment and identification. The identification phase includes separate
641 pre-search feature extraction stage, and a search stage.642 The design reflects the following *testing* objectives for 1:N implementations.

- support distributed enrollment on multiple machines, with multiple processes running in parallel
- allow recovery after a fatal exception, and measure the number of occurrences
- allow NIST to copy enrollment data onto many machines to support parallel testing
- respect the black-box nature of biometric templates
- extend complete freedom to the provider to use arbitrary algorithms
- support measurement of duration of core function calls
- support measurement of template size

Table 23 – Procedural overview of the identification test

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	<p>Give the implementation advance notice of the number of individuals and images that will be enrolled.</p> <p>Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty.</p> <p>The implementation is permitted read-write-delete access to the enrollment directory during this phase. The implementation is permitted read-only access to the configuration directory.</p> <p>After enrollment, NIST may rename and relocate the enrollment directory - the implementation should not depend on the name of the enrollment directory.</p>	
	E2	Parallel Enrollment	<p>For each of N individuals, pass multiple images of the individual to the implementation for conversion to a combined template. The implementation will return a template to the calling application.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase. NIST's calling application will be responsible for storing all templates as binary files. These will not be available to the implementation during this enrollment phase.</p> <p>Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. The same person will not be enrolled twice.</p>	<p>Statistics of the times needed to enroll an individual.</p> <p>Statistics of the sizes of created templates.</p> <p>The incidence of failed template creations.</p>
	E3	Finalization	<p>Permanently finalize the enrollment directory. This supports, for example, adaptation of the image-processing functions, adaptation of the representation, writing of a manifest, indexing, and computation of statistical information over the enrollment dataset.</p> <p>The implementation is permitted read-write-delete access to the enrollment directory during this phase.</p>	<p>Size of the enrollment database as a function of population size N and the number of images.</p> <p>Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.</p>
Pre-search	S1	Initialization	<p>Tell the implementation the location of an enrollment directory. The implementation could look at the enrollment data.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase. Statistics of the time needed for this operation.</p>	Statistics of the time needed for this operation.
	S2	Template preparation	<p>For each probe, create a template from a set of input images. This operation will generally be conducted in a separate process invocation to step S2.</p> <p>The implementation is permitted no access to the enrollment directory during this phase.</p> <p>The result of this step is a search template.</p>	<p>Statistics of the time needed for this operation.</p> <p>Statistics of the size of the search template.</p>
Search	S3	Initialization	<p>Tell the implementation the location of an enrollment directory. The implementation should read all or some of the enrolled data into main memory, so that searches can commence.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase.</p>	Statistics of the time needed for this operation.
	S4	Search	<p>A template is searched against the enrollment database.</p> <p>The implementation is permitted read-only access to the enrollment directory during this phase.</p>	<p>Statistics of the time needed for this operation.</p> <p>Accuracy metrics - Type I + II error rates.</p>

3.4.2. Initialization of the enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization function of Table 24.

Table 24 – Enrollment initialization

Removed fields: num_descriptions

Prototype	int32_t initialize_enrollment_session(const string &configuration_location, const string &enrollment_directory, const uint32_t num_persons, const uint32_t num_images, const std::vector<string> &descriptions);	Input Input Input Input Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any $M \geq 1$ calls to convert MULTIFACE _to_enrollment_template. The SDK should tolerate execution of $P > 1$ processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	num_persons	The number of persons who will be enrolled $0 \leq N \leq 2^{32} - 1$ (e.g. 1million)
	num_images	The total number of images that will be enrolled, summed over all identities $0 \leq M \leq 2^{32} - 1$ (e.g. 1.8 million)
	descriptions	A lexicon of labels one of which will be assigned to each enrollment image. EXAMPLE: The descriptions could be {"mugshot", "visa"}. NOTE: The identification search images may or may not be labeled. An identification image may carry a label not in this set of labels. The number of items stored in the vector is accessible via the vector::size() function.
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission, space).
	6	The SDK cannot support the number of persons or images.
	8	The descriptions are unexpected, or unusable.
	Other	Vendor-defined failure

3.4.3. Enrollment

A **MULTIFACE** is converted to a single enrollment template using the function of Table 25.

Table 25 – Enrollment feature extraction

Prototypes	int32_t convert_ MULTIFACE _to_enrollment_template(const MULTIFACE &input_faces, std::vector<EYEPAIR> &output_eyes, uint32_t &template_size, uint8_t *proprietary_template);	Input Output Output Output
Description	This function takes a MULTIFACE , and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. If the function executes correctly (i.e. returns a zero exit status), the NIST calling application will store the template.	

	<p>The NIST application will concatenate the templates and pass the result to the enrollment finalization function (see section 3.4.4).</p> <p>If the function gives a non-zero exit status:</p> <ul style="list-style-type: none"> – If the exit status is 8, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. <p>IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function of section 3.4.4.</p>	
Input Parameters	input_faces	An instance of a Table 13 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	output_eyes	For each input image in the MULTIFACE the function shall return the estimated eye centers. The calling application will pre-allocate the correct number of EYEPAIR structures (i.e. one for each image in the MULTIFACE).
	template_size	The size, in bytes, of the output template
	proprietary_template	The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the MULTIFACE ; the value T is output by the maximum enrollment template size function of Table 18.
Return Value	0	Success
	2	Elective refusal to process this kind of MULTIFACE
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

652

653 3.4.4. Finalize enrollment

654 After all templates have been created, the function of Table 26 will be called. This freezes the enrollment data. After this
 655 call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search
 656 phases.

657 The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and
 658 data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data.
 659 No output is expected from this function, except a return code.

660

Table 26 – Enrollment finalization

Prototypes	int32_t finalize_enrollment (
	const string &enrollment_directory,	Input
	const string &edb_name,	Input
	const string &edb_manifest_name);	Input
Description	<p>This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.4. The enrollment directory permissions will be read + write.</p> <p>The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.</p> <p>This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.</p>	
Input Parameters	enrollment_directory	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edb_name	The name of a single file containing concatenated templates, i.e. the EDB of section 2.4.

		While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edb_manifest_name	The name of a single file containing the EDB manifest of section 2.4. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
Return Value	0	Success
	2	Cannot locate the input data - the input files or names seem incorrect.
	4	An operation on the enrollment directory failed (e.g. permission, space).
	6	One or more template files are in an incorrect format.
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

661 3.4.5. Pre-search feature extraction

662 3.4.5.1. Initialization

663 Before **MULTIFACE**s are sent to the identification feature extraction function, the test harness will call the initialization
664 function in Table 27.

665 **Table 27 – Identification feature extraction initialization**

Prototype	int32_t initialize_feature_extraction_session(const string &configuration_location, const string &enrollment_directory);	
		Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before any $M \geq 1$ calls to convert_ MULTIFACE _to_identification_template. The SDK should tolerate execution of $P \Rightarrow 1$ processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	Input
		Input
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission).
	Other	Vendor-defined failure

666 3.4.5.2. Feature extraction

667 A **MULTIFACE** is converted to an atomic identification template using the function of Table 28. The result may be stored
668 by NIST, or used immediately. The SDK shall not attempt to store any data.

669 **Table 28 – Identification feature extraction**

Prototypes	int32_t convert_ MULTIFACE _to_identification_template(const MULTIFACE &input_faces, std::vector<EYEPAIR> &output_eyes, uint32_t &template_size,	
		Input
		Output
		Output

	uint8_t *identification_template);	Output
Description	<p>This function takes a MULTIFACE, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result.</p> <p>If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations.</p> <p>The function shall not have access to the enrollment data, nor shall it attempt access.</p>	
Input Parameters	input_faces	An instance of a Table 13 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	output_eyes	For each input image in the MULTIFACE the function shall return the estimated eye centers. The calling application will pre-allocate the correct number of EYEPAIR structures (i.e. one for each image in the MULTIFACE).
	template_size	The size, in bytes, of the output template
	identification_template	The output template for a subsequent identification search. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the input MULTIFACE ; the value T is output by the maximum enrollment template size function of Table 18.
Return Value	0	Success
	2	Elective refusal to process this kind of MULTIFACE
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

670 3.4.6. Initialization

671 The function of Table 29 will be called once prior to one or more calls of the searching function of Table 30. The function
672 might set static internal variables so that the enrollment database is available to the subsequent identification searches.

673 **Table 29 – Identification initialization**

Prototype	int32_t initialize_identification_session(
	const string &configuration_location,	Input
	const string &enrollment_directory);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the finalize_enrollment function.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed.
Return Value	0	Success
	Other	Vendor-defined failure

674 3.4.7. Search

675 The function of Table 30 compares a proprietary identification template against the enrollment data and returns a
676 candidate list.

677 **Table 30 – Identification search**

Prototype	int32_t identify_template(
	const uint8_t *identification_template,	Input
	const uint32_t identification_template_size,	Input
	const uint32_t candidate_list_length,	Input
	std::vector<CANDIDATE> &candidate_list,	Output

	bool &decision);	Output
Description	This function searches a template against the enrollment set, and outputs a list of candidates. NIST will pre-allocate the vector with candidates before the call.	
Input Parameters	identification_template	A template from convert_MULTIFACE_to_identification_template() - If the value returned by that function was non-zero the contents of identification_template will not be used and this function (i.e. identify_template) will not be called.
	identification_template_size	The size, in bytes, of the input identification template $0 \leq N \leq 2^{32} - 1$
	candidate_list_length	The number of candidates the search should return
Output Parameters	candidate_list	A vector containing "candidate_list_length" objects of candidates. The datatype is defined in section 2.5. Each candidate shall be populated by the implementation. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
	decision	A best guess at whether there is a mate within the enrollment database. If there was a mate found, this value should be set to true, Otherwise, false. Many such decisions allow a single point to be plotted alongside a DET
Return Value	0	Success
	2	The input template was defective.
	Other	Vendor-defined failure

678

679 NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \leq L \leq$
680 200, and $L \ll N$. However, there is interest in the presence of mates much further down the candidate list. We may
681 therefore extend the candidate list length such that L approaches N.

682 3.5. Pose conformance, age, gender, and expression neutrality estimation

683 The MEDS database¹¹ includes many facial images for which age and gender are provided. The FERET database does
684 likewise¹². It also includes images for which the non-frontal pose is known. A number of academic databases do likewise:
685 For example the CMU PIE databases famously include pose illumination and expression variation¹³.

686 3.5.1. Pose conformance

687 The functions of this section support testing whether a face in an image has frontal pose. This supports conformance
688 testing of, for example, the Full Frontal specification of the ISO standard [ISO]. The goal is to support a marketplace of
689 products for acquisition time assessment of pose. This is important because pose is arguably the most influential
690 covariate on face recognition error rates, and is not generally controllable by design of the acquisition system. This
691 problem has been investigated in literature¹⁴.

692 NIST encourages participants in this study to implement real-time video rate implementations, and also slower more
693 accurate methods.

694 The functional specification here supports a DET analysis in which false-rejection of actually frontal images can be traded
695 off against false acceptance of non-frontal images via a frontal-conformance parameter, t. The exact meaning of the
696 "frontality" value returned by this function is not regulated by the NIST specification. However a reasonable
697 implementation would embed a monotonic relationship between the output value and non-frontal angle (i.e. compound
698 rotation involving azimuthal head yaw and pitch).

699 The formal ISO requirement is for five degree rotation in pitch and yaw. While the ISO standard establishes an eight
700 degree limit on roll angle, this is of less importance. NIST will not consider roll angle.

¹¹ The Multiple Encounter Deceased Subject Database, NIST Special Database 32, is freely available here:
<http://www.nist.gov/itl/iad/ig/sd32.cfm>

¹² FERET is available via a different process here: <http://www.nist.gov/itl/iad/ig/feret.cfm>

¹³ For example, the CMU Multi-PIE Face Database – <http://www.multipie.org/> and others

¹⁴ Erik Murphy-Chutorian and Mohan Manubhai Trivedi, "Head Pose Estimation in Computer Vision: A Survey," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 31, no. 4, pp. 607-626, 2009.

701 3.5.2. Age

702 The functions of this section support estimation of the age of a face in an image. The process of age determination has
703 potential application in at least the following areas:

- 704 – Age-based access control
- 705 – Age adaptive human machine interaction (e.g. marketing)
- 706 – Age invariant person identification
- 707 – Data mining and organization

708
709 Age estimation¹⁵ has its own set of unique challenges when compared to other face image interpretation tasks, including
710 limited inter-age group variation especially when dealing with mature subjects, diversity of aging variation between races
711 and gender, and dependence on external factors such as health conditions, lifestyle, cosmetic surgery, etc.

712 3.5.3. Gender

713 The functions of this section support estimation of the gender¹⁶ of a face in an image. Similar to age, gender is viewed as
714 a soft biometric trait that has applications in surveillance, human-computer interaction and image retrieval systems.
715 Gender could potentially be leveraged to index biometric databases and enhance the recognition accuracy of primary
716 traits such as face.

717 3.5.4. Expression Neutrality

718 Facial expression recognition is an important aspect in interpersonal communication and human-machine interaction,
719 having applications, for example, in building intelligent and more intuitive human-machine interfaces. ISO/IEC 19794-
720 5:2005 establishes codes for facial expression. Clause 5.5.7 of that standard defines a neutral expression as “(non-smiling)
721 with both eyes open and mouth closed”.

722 3.5.5. API

723 Vendors may submit a class D SDK to evaluate performance on estimation of pose conformance, age, gender, and/or
724 expression neutrality. The SDK must define a C++ class named exactly SdkEstimator, which subclasses from the Estimator
725 class (see Table 31). At a minimum, the developer’s SdkEstimator class must override at least one of the estimation
726 functions and its corresponding initialization function from Table 31. To support those who only want to implement a
727 subset of the class D estimation functions, any functions that are not overridden by the developer’s SDK will default to the
728 behavior specified in the “Base” Estimator Class (ie. return a value indicating function is “not implemented”).
729

730 **Table 31 – “Base” Estimator Class Structure**

	C++ code fragment	Remarks
1.	<code>#include <vector></code> <code>#include <string></code>	
2.	<code>class Estimator {</code>	
3.	<code>public:</code>	
4.	<code>virtual ~Estimator();</code>	
5.	<code>virtual int32_t initialize_frontal_pose_estimation(const std::string &configuration_location);</code>	Pose conformance estimation initialization
6.	<code>virtual int32_t estimate_frontal_pose_conformance(const ONEFACE &input_face, double &non_frontality);</code>	Pose conformance estimation

¹⁵ Xin Geng, Zhi-Hua Zhou, and Kate Smith-Miles, “Automatic Age Estimation Based on Facial Aging Patterns,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 29, no. 12, pp. 2234-2240, 2007.

¹⁶ C.H. Ting, U.U. Sheikh, and S.A.R. Abu-Bakar, “Gender estimation based on physiological features of the face”, 10th International Conference on Information Science, ISSPA, pp. 201-204, 2010.

7.	<code>virtual int32_t initialize_age_estimation(const std::string &configuration_location);</code>	Age estimation initialization
8.	<code>virtual int32_t estimate_age(const ONEFACE &input_face, int32_t &age);</code>	Age (in years) estimation
9.	<code>virtual int32_t initialize_gender_estimation(const std::string &configuration_location);</code>	Gender estimation initialization
10.	<code>virtual int32_t estimate_gender(const ONEFACE &input_face, int8_t &gender, double &mf);</code>	Gender estimation
13.	<code>virtual int32_t initialize_expression_estimation(const std::string &configuration_location);</code>	Expression neutrality estimation initialization
14.	<code>virtual int32_t estimate_expression_neutrality(const ONEFACE &input_face, double &expression_neutrality);</code>	Expression neutrality estimation
15.	<code>};</code>	

731

732

733

734

735

736

737

An example of how the SdkEstimator class could be implemented is provided in Table 32 and Table 33. In the example, the pose estimation function and its corresponding initialization function are implemented. In this case, during runtime, the developer implementation of pose estimation will be executed. The rest of the unimplemented functions will default to the behavior specified in the “Base” Estimator class (see Table 31).

Table 32 – Example of SdkEstimator Class Declaration

	C++ code fragment – sdkestimator.h	Remarks
1.	<code>#include <frvt2012.h></code>	
2.	<code>class SdkEstimator : public Estimator {</code>	
3.	<code>public:</code>	
4.	<code> SdkEstimator();</code>	Default constructor
5.	<code> ~SdkEstimator();</code>	Default destructor
6.	<code> int32_t initialize_frontal_pose_estimation(const std::string &configuration_location);</code>	Pose conformance estimation initialization
7.	<code> int32_t estimate_frontal_pose_conformance(const ONEFACE &input_face, double &non_frontality);</code>	Pose conformance estimation
8.	<code>};</code>	

738

739

740

Table 33 – Example of SdkEstimator Class Definition

	C++ code fragment – sdkestimator.cpp	Remarks
1.	<code>#include <sdkestimator.h></code>	
2.	<code>SdkEstimator::SdkEstimator() { }</code>	Default constructor
3.	<code>SdkEstimator::~SdkEstimator() { }</code>	Default destructor
4.	<code>int32_t SdkEstimator::initialize_frontal_pose_estimation(const std::string &configuration_location) { return 0; }</code>	Override the pose conformance estimation initialization function

5.	<pre> int32_t SdkEstimator::estimate_frontal_pose_conformance(const ONEFACE &input_face, double &non_frontality) { non_frontality = 0.1; return 0; } </pre>	Override the pose conformance estimation function
6.	};	

741

742

743

744

745

746

The initialization functions of Table 34 will be called before one or more calls to the corresponding pose conformance, age, gender, and expression neutrality estimation functions. In other words, `initialize_frontal_pose_estimation()` will be called prior to `estimate_frontal_pose_conformance()`, `initialize_age_estimation()` will be called prior to `estimate_age()`, `initialize_gender_estimation()` will be called prior to `estimate_gender()`, and `initialize_expression_estimation()` will be called prior to `estimate_expression_neutrality()`.

747

Table 34 – Initialization of Pose conformance, Age, Gender, and Expression neutrality estimation

Prototypes	<code>int32_t initialize_frontal_pose_estimation(</code>	
	<code>const string &configuration_location);</code>	Input
	<code>int32_t initialize_age_estimation(</code>	
	<code>const string &configuration_location);</code>	Input
	<code>int32_t initialize_gender_estimation(</code>	
	<code>const string &configuration_location);</code>	Input
	<code>int32_t initialize_expression_estimation(</code>	
	<code>const string &configuration_location);</code>	Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any corresponding call to the Table 35 functions. The SDK under test should set all parameters.	
Input Parameters	<code>configuration_location</code>	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
Output Parameters	<code>none</code>	
Return Value	<code>0</code>	Success
	<code>2</code>	Vendor provided configuration files are not readable in the indicated location.
	<code>Other</code>	Vendor-defined failure

748

749

750

751

Table 35 provides more details on the functions for computing a pose conformance, age, gender, and expression neutrality from an image.

752

Table 35 – Pose conformance, Age, Gender, Expression neutrality estimation

	<code>int32_t estimate_frontal_pose_conformance(</code>	Input
	<code>const ONEFACE &input_face,</code>	Output
	<code>double &non_frontality);</code>	
Prototypes	<code>int32_t estimate_age(</code>	
	<code>const ONEFACE &input_face,</code>	Input
	<code>int32_t &age);</code>	Output
	<code>int32_t estimate_gender(</code>	
	<code>const ONEFACE &input_face,</code>	Input
	<code>int8_t &gender</code>	Output
	<code>double &mf);</code>	Output
	<code>int32_t estimate_expression_neutrality(</code>	
	<code>const ONEFACE &input_face,</code>	Input

	double &expression_neutrality);	Output
Descriptions	<p>estimate_frontal_pose_conformance - this function takes a ONEFACE, and outputs a non-frontality value for the image. The non-frontality value should increase with larger deviations from frontal pose.</p> <p>estimate_age – this function takes a ONEFACE, and outputs an age value (in years) for the image.</p> <p>estimate_gender - this function takes a ONEFACE, and outputs a gender value and a maleness-femaleness value for the image.</p> <p>estimate_expression_neutrality – this function takes a ONEFACE, and an expression neutrality value for the image.</p>	
Input Parameters	input_face	An instance of a Table 12 structure.
Output Parameters	non-frontality	Indication of how far from frontal the head pose is. The value should be on the range [0,1].
	age	Indication of the age (in years) of the person. The value should be on the range [0,100].
	gender	Indication of the gender of the person. Valid values are 0: Male 1: Female -1: Unknown
	mf	A real-valued measure of maleness-femaleness value on [0,1]. A value of 0 indicates certainty that the subject is a male, and a value of 1 indicates certainty that the subject is a female.
	expression_neutrality	ISO/IEC 19794-5:2005 establishes codes for facial expression. Clause 5.5.7 of that standard defines a neutral expression as “(non-smiling) with both eyes open and mouth closed”. SDKs shall report a real-valued measure of expression neutrality on [0,1] with 0 denoting large deviation from neutral and 1 indicating a fully neutral expression.
Return Value	0	Success
	2	Elective refusal to process this kind of ONEFACE
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

NOTE 1 The "mf" and "non-frontality" values can be used to make DET characteristics. These would plot, respectively, the "False male rate vs. False female rate" for gender, and the "False non-frontal rate vs. False frontal rate" for pose. Various summary statistics can be computed also.

3.6. Video

3.6.1. Definitions

As shown in Table 36, the video API supports 1:N identification of video-to-video, video-to-still image, and still image-to-video. The following hold:

- A still image is a picture of one and only one person. One or more such images are presented to the implementation using a **MULTIFACE** data structure
- A video is a sequence of $F \geq 1$ frames containing $P \geq 0$ persons.
- A frame is 2D still image containing $P \geq 0$ persons
- Any person might be present in $0 \leq f \leq F$ frames, and their presence may be non-contiguous (e.g. due to occlusion)
- Different videos contain different numbers of frames and people.
- A **ONEVIDEO** container is used to represent a video. It contains a small header and pointers to F frames.
- Any person found in a video is represented by proprietary template (feature) data contained with a **PERSONREP** data structure. A proprietary template contains information from one or more frames. Internally, it might embed multiple traditional still-image templates, or it might integrate feature data by tracking a person across multiple frames.
- A **PERSONREP** structure additionally contains a trajectory indicating the location of the person in each frame.

Please note that all of the code for the classes needed to implement the video API will be provided to implementers at <http://nigos.nist.gov:8080/frvt2012/>. A single sample video has been made available at the same link. The sample video is only approximately representative of the scene and is not an extraction from the actual video data that will be used in the evaluation. It is only intended to illustrate similarities in terms of camera placement relative to the subject and people behavior. It is not intended to represent the optical properties of the actual imaging systems, particularly the spatial sampling rate, nor the compression characteristics. More information will be released moving forward.

NIST does not know the minimum and maximum numbers of persons appearing in video sequences. Moreover, NIST will apply the algorithms to other databases. The maximum number of frames in a video sequence will be limited by the duration of the sequence. NIST expects to use sequences whose duration extends from a few seconds to a few minutes.

NIST does not anticipate using interlaced video.

The frame sizes will often be 1920 x 1080 pixels. We do not anticipate using larger sizes.

The videos are contiguous in time, without interruptions.

Much of the video data is present at 30 frames per second.

Some sequences exist at much higher frame rates. NIST will examine whether this offers benefit.

Much of the data was collected using a modern proprietary video codec intended to allow inspection of faces.

In the videos, the scenes capture people walking towards the camera. Occasionally, there are people walking in various transverse directions including people walking away from the camera. The cameras have varying pitch angles ranging from 0 degrees (frontal) to higher values. The depth of scene varies between the cameras such that the sizes of the faces vary, with the following:

- Eye-to-eye distances range from approximately 40 pixels to 120 pixels
- Amount of time a face is fully visible in a scene can vary from approximately 0 to 5 seconds
- Some of the captures include non-uniform lighting due to light coming through adjacent windows.

Table 36 – API implementation requirements for Video

Function	Video-to-video	Still-to-video	Video-to-still
Enroll	Videos	Videos	Stills
Enrollment input datatype	ONEVIDEO	ONEVIDEO	MULTIFACE

Enrollment datatype	PERSONREP	PERSONREP	PERSONREP
Search	Video	Still	Video
Search input datatype	ONEVIDEO	MULTIFACE	ONEVIDEO
Search datatype	PERSONREP	PERSONREP	PERSONREP
Search result	CANDIDATELIST	CANDIDATELIST	CANDIDATELIST
API requirements	3.6.9 + 3.6.10 + 3.6.12 + 3.6.14	3.6.9 + 3.6.10 + 3.6.20 + 3.6.14	3.6.16 + 3.6.18 + 3.6.12 + 3.6.21

805 3.6.1.1. Video-to-video

806 Video-to-video identification is the process of enrolling N videos and then searching the enrollment database with a
807 search video. During identification, the SDK shall return a set of indices of candidate videos that contain people who
808 appear in the search video.

- 809 – N templates will be generated from M enrollment videos. If no people appear in the videos, N will be 0. If may
810 people appear in each video, we'd expect $N > M$.
- 811 – The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- 812 – A ONEVIDEO will be converted to $S \geq 0$ identification template(s) based on the number of people detected in the
813 video.
- 814 – Each identification template generated will be searched against the enrollment database of templates generated
815 from the M input videos.

816 NOTE 1 We anticipate that the same person may appear in more than one enrolled video.

818 3.6.1.2. Still image-to-video

819 Still image-to-video identification is the process of enrolling N videos and then searching the enrollment database with a
820 template produced from a MULTIFACE as follows:

- 821 – N templates will be generated from $1 < M \leq N$ enrollment videos.
- 822 – The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- 823 – A MULTIFACE (still image) will be converted to an identification template.
- 824 – The identification template will be searched against the enrollment database of N templates.

825 NOTE 1 We anticipate that the same person may appear in more than one enrolled video.

828 3.6.1.3. Video-to-still image

829 Video-to-still image identification is the process of enrolling N MULTIFACES (see Table 13) and then searching the
830 enrollment database with templates from persons found in a video as follows

- 831 – N templates will be generated from N still-image MULTIFACES.
- 832 – The N templates will be concatenated and finalized into a proprietary enrollment data structure.
- 833 – A ONEVIDEO will be converted to $S \geq 0$ identification template(s) based on the number of people detected in the
834 video.
- 835 – Each of the S identification templates will be searched separately against the enrollment database of N templates.

836 3.6.2. Class for encapsulating a video sequence

837 Table 37 – ONEVIDEO Class

	C++ code fragment	Remarks
1.	class ONEVIDEO	
2.	{ private:	
3.	uint16_t frameWidth;	Number of pixels horizontally of all frames

4.	uint16_t frameHeight;	Number of pixels vertically of all frames
5.	uint8_t frameDepth;	Number of bits per pixel for all frames. Legal values are 8 and 24.
6.	uint16_t framesPerSec;	The frame rate of the video sequence in seconds
7.	public: std::vector<uint8_t*> data;	Vector of pointers to data from each frame in the video sequence. The number of frames (ie. size of the vector) can be obtained by calling vector::size(). The i-th entry in data (ie. data[i]) points to frame_width x frame_height pixels of data for the i-th frame.
8.	//Getter and Setter Methods	
9.	};	

838 3.6.3. Class representing a pair of eye coordinates

839 The data structure for reporting person locations in video appears in Table 38. The coordinates may be useful to NIST for
840 relating spatial location to recognition success during our analysis.

841 Table 38 – EYEPAIR Class

	C++ code fragment	Remarks
1.	class EYEPAIR	
2.	{	
3.	private:	
3.	bool isSet;	If the eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise it should be set to false.
4.	int16_t xLeft; int16_t yLeft;	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
5.	int16_t xRight; int16_t yRight;	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
6.	uint16_t frameNum	For video: the frame number that corresponds to the video frame from which the eye coordinates were generated. (ie. the i-th frame from the video sequence). This field should not be set for eye coordinates for a single still image.
7.	public: //getter/setter methods	
8.	};	

842 3.6.4. Data type for representing a person's trajectory via eye coordinates from a video sequence

843 Table 39 – PersonTrajectory typedef

	C++ code fragment	Remarks
1.	typedef std::vector<EYEPAIR> PersonTrajectory;	Vector of EYEPAIR (see 3.6.3) objects for video frames where eyes were detected. This data structure should store eye coordinates for each video frame where eyes were detected for a particular person. For video frames where the person's eyes were not detected, the SDK shall not add an EYEPAIR to this data structure. If a face can be detected, but not the eyes, this structure could be populated with (x,y) _{LEFT} == (x,y) _{RIGHT}

844 3.6.5. Class for representing a person from a video sequence or an image

845 Table 40 – PERSONREP Class

	C++ code fragment	Remarks
1.	class PERSONREP	
2.	{	
3.	private: PersonTrajectory eyeCoordinates;	Data structure for capturing eye coordinates

4.	<code>PersonTemplate proprietaryTemplate;</code>	PersonTemplate is a wrapper to a <code>uint8_t*</code> for capturing proprietary template data representing a person from a video sequence or an image.
5.	<code>public:</code>	
6.	<code>PERSONREP(const uint64_t inSize);</code>	The constructor takes a size parameter and allocates memory of <i>inSize</i> . <code>getPersonTemplatePtr()</code> should be called to access the newly allocated memory for SDK manipulation. Please note that this class will take care of all memory allocation and de-allocation of its own memory. The SDK shall not de-allocate memory created by this class.
7.	<code>void pushBackEyeCoord(const EYEPAIR &eyes);</code>	This function should be used to add EYEPAIRS for the video frames or images where eye coordinates were detected.
8.	<code>uint8_t* getPersonTemplatePtr() const;</code>	This function returns a <code>uint8_t*</code> to the template data.
9.	<code>uint64_t getPersonTemplateSize() const;</code>	This function returns the size of the template data.
10.	<code>//... getter methods, copy constructor,</code>	
11.	<code>//... assignment operator</code>	
12.	<code>};</code>	

846 3.6.6. Class for result of an identification search

847 All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most
848 similar matching entries list first with lowest rank.

849 **Table 41 – CANDIDATE Class**

	C++ code fragment	Remarks
1.	<code>class CANDIDATE</code>	
2.	<code>{</code>	
	<code>private:</code>	
3.	<code>bool isSet</code>	If the candidate is valid, this should be set to true. If the candidate computation failed, this should be set to false.
4.	<code>uint32_t templateId;</code>	The Template ID integer from the enrollment database manifest defined in clause 0.
5.	<code>double similarityScore;</code>	Measure of similarity between the identification template and the enrolled candidate. Higher scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	<code>public:</code>	
	<code>//getter/setter methods</code>	
7.	<code>};</code>	

850 3.6.7. Data type for representing a list of results of an identification search

851 **Table 42 – CANDIDATELIST typedef**

	C++ code fragment	Remarks
1.	<code>typedef std::vector<CANDIDATE> CANDIDATELIST;</code>	A vector containing objects of CANDIDATE s. The CANDIDATE class is defined in section 3.6.6.

853 3.6.8. Class representing return code values

854 **Table 43 – ReturnCode class**

	C++ code fragment	Remarks
	<code>class ReturnCode {</code>	
	<code>public:</code>	
1.	<code>enum Status</code>	

2.	{	
3.	Success=0,	Success
4.	MissingConfig=1,	The configuration data is missing or unreadable
5.	EnrollDirFailed=2,	An operation on the enrollment directory failed
6.	InitNumData=3,	The SDK can't support the number of images or videos
7.	InitBadDesc=4,	The image descriptions are unexpected or unusable
8.	RefuseInput=5,	Elective refusal to process this kind of input (ONEVIDEO or MULTIFACE)
9.	FailExtract=6,	Involuntary failure to extract features
10.	FailTempl=7,	Elective refusal to produce a template
11.	FailParse=8,	Cannot parse input data
12.	FinInputData=9,	Cannot locate input data
13.	FinTemplFormat=10,	One or more template files are in an incorrect format
14.	IdBadTempl=11,	The input template was defective
15.	Vendor=88	Vendor-defined failure
16.	};	
17.	ReturnCode(const Status inStatus);	Constructor that takes an input parameter of a Status enum value. All of the functions that need to be implemented for the Video API return an instantiation of a ReturnCode object with a valid status value passed in as a parameter.
18.	Status getStatus() const;	Getter method to return status value
19.	private:	
20.	Status status;	Member variable for storing status
21.	};	

855 3.6.9. The VideoEnrollment Interface

856 The abstract class VideoEnrollment must be implemented by the SDK developer in a class named exactly
857 SdkVideoEnrollment. The processing that takes place during each phase of the test is done via calls to the methods
858 declared in the interface as pure virtual, and therefore is to be implemented by the SDK. The test driver will call these
859 methods, handling all return values.

	C++ code fragment	Remarks
1.	class VideoEnrollment	
2.	{	
	public:	
3.	virtual ReturnCode getPid(string &sdkId, string &email) = 0;	Return the sdk identifier and email
4.	virtual ReturnCode initialize(const string &configDir, const string &enrollDir, const uint32_t numVideos) = 0 ;	Initialize the enrollment session.
5.	virtual ReturnCode generateEnrollmentTemplate(const ONEVIDEO &inputVideo, vector< PERSONREP > &enrollTemplates) = 0;	Generate enrollment template(s) for the persons detected in the input video. This function takes an ONEVIDEO (see 3.6.2) as input and populates a vector of PERSONREP (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.
6.	// Destructor	
7.	};	

860 3.6.9.1. Implementation identifier

861 **Table 44 – VideoEnrollment::getPid**

Prototype	ReturnCode getPid(string &sdkId,	
		A developer-assigned ID. This shall be different for each submitted SDK.

	string &email);	Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdkId	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email	Point of contact email address. The value cannot be the empty string.
ReturnCode	Success	Success
	Vendor	Vendor-defined failure

862 3.6.9.2. Initialization of the video enrollment session

863 Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-
864 to-video and still image-to-video.

865 **Table 45 – VideoEnrollment::initialize**

Prototype	ReturnCode initialize(const string &configDir, const string &enrollDir, const uint32_t numVideos);	
		Input
		Input
		Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any $M \geq 1$ calls to generateEnrollmentTemplate. The SDK should tolerate execution of $P > 1$ processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	numVideos	The total number of videos that will be passed to the SDK for enrollment.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	InitNumData	The SDK cannot support the number of videos.
	Vendor	Vendor-defined failure

866 3.6.9.3. Video enrollment

867 An **ONEVIDEO** is converted to enrollment template(s) for each person detected in the **ONEVIDEO** using the function
868 below.

869 **Table 46 – VideoEnrollment::generateEnrollmentTemplate**

Prototypes	ReturnCode generateEnrollmentTemplate(const ONEVIDEO &inputVideo, std::vector< PERSONREP > &enrollTemplates);	
		Input
		Output
Description	This function takes an ONEVIDEO , and outputs a vector of PERSONREP objects. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function. For a video in which no persons appear, a valid output is an empty vector (i.e. size() == 0). If the function gives a non-zero exit status:	

	<ul style="list-style-type: none"> – If the exit status is <code>ReturnCode::FailParse</code>, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. <p>IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.</p>	
Input Parameters	inputVideo	An instance of a Table 37 class.
Output Parameters	enrollTemplates	For each person detected in the ONEVIDEO , the function shall identify the person's estimated eye centers for each video frame where the person's eye coordinates can be calculated. The eye coordinates shall be captured in the PERSONREP .eyeCoordinates variable, which is a vector of EYEPAIR objects. The frame number from the video of where the eye coordinates were detected shall be captured in the EYEPAIR .frameNum variable for each pair of eye coordinates. In the event the eye centers cannot be calculated (ie. the person becomes out of sight for a few frames in the video), the SDK shall not store an EYEPAIR for those frames.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of ONEVIDEO
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

870 3.6.10. The VideoFinalize Interface

871 The abstract class VideoFinalize must be implemented by the SDK developer in a class named exactly SdkVideoFinalize.
872 The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its
873 manifest have been stored. These are described in section 2.4. The enrollment directory permissions will be read +
874 write.

	C++ code fragment	Remarks
1.	class VideoFinalize	
2.	{	
	public:	
3.	virtual ReturnCode finalize(const string &enrollDir, const string &edbName, const string &edbManifest) = 0;	This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.
4.	// Destructor	
5.	};	

875 3.6.11. Finalize video enrollment

876 After all templates have been created, the function of Table 47 will be called. This freezes the enrollment data. After this
877 call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search
878 phases.

879 The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and
880 data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data.
881 No output is expected from this function, except a return code.

882 **Table 47 – VideoFinalize::finalize**

Prototypes	ReturnCode finalize (
	const string &enrollDir,	Input

	const string &edbName, const string &edbManifest);	Input Input
Description	<p>This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.4. The enrollment directory permissions will be read + write.</p> <p>The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.</p> <p>This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.</p>	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 2.4. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 2.4. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

883 3.6.12. The VideoFeatureExtraction Interface

884 The abstract class VideoFeatureExtraction must be implemented by the SDK developer in a class named exactly
885 SdkVideoFeatureExtraction.

	C++ code fragment	Remarks
1.	class VideoFeatureExtraction	
2.	{	
	public:	
3.	virtual ReturnCode initialize(const string &configDir, const string &enrollDir) = 0;	Initialize the feature extraction session.
4.	virtual ReturnCode generateIdTemplate(const ONEVIDEO &inputVideo, vector<PERSONREP> &idTemplates) = 0;	Generate identification template(s) for the persons detected in the input video. This function takes an ONEVIDEO (see 3.6.2) as input and populates a vector of PERSONREP (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.
5.	// Destructor	
6.	};	

886 3.6.13. Video feature extraction initialization

887 Before one or more ONEVIDEOS are sent to the identification feature extraction function, the test harness will call the
888 initialization function below.

889 Table 48 – VideoFeatureExtraction::initialize

Prototype	ReturnCode initialize(const string &configDir,	
		Input

	const string &enrollDir);	Input
Description	<p>This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before any $M \geq 1$ calls to generateIdTemplate. The SDK should tolerate execution of $P \Rightarrow 1$ processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel.</p> <p>The implementation has read-only access to its prior enrollment data.</p>	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

890 3.6.13.1. Video feature extraction

891 An **ONEVIDEO** is converted to one or more identification templates using the function below. The result may be stored by
 892 NIST, or used immediately. The SDK shall not attempt to store any data.

893 **Table 49 – VideoFeatureExtraction::generateIdTemplate**

Prototypes	ReturnCode generateIdTemplate(const ONEVIDEO &inputVideo, std::vector< PERSONREP > &idTemplates);	
		Input Output
Description	<p>This function takes an ONEVIDEO (see 3.6.2) as input and populates a vector of PERSONREP (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.</p> <p>If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations.</p> <p>The function shall not have access to the enrollment data, nor shall it attempt access.</p>	
Input Parameters	InputVideo	An instance of a section 3.6.2 class. Implementations must alter their behavior according to the people detected in the video sequence.
Output Parameters	IdTemplates	For each person detected in the video, the function shall create a PERSONREP (see section 3.6.5) object, populate it with a template and eye coordinates for each frame where eyes were detected, and add it to the vector.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of ONEVIDEO
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

894 3.6.14. The VideoSearch Interface

895 The abstract class VideoSearch must be implemented by the SDK developer in a class named exactly SdkVideoSearch.

	C++ code fragment	Remarks
1.	class VideoSearch	

2.	{	
3.	public:	
	virtual ReturnCode initialize(const string &configDir, const string &enrollDir) = 0;	Initialize the search session.
4.	virtual ReturnCode identifyVideo(const PERSONREP &idVideoTemplate, const uint32_t candListLength, CANDIDATELIST &candList) = 0;	For video-to-video identification This function searches a template generated from an ONEVIDEO against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7).
5.	virtual ReturnCode identifyImage(const PERSONREP &idImageTemplate, const uint32_t candListLength, CANDIDATELIST &candList) = 0;	For still-to-video identification This function searches a template generated from a MULTIFACE against the enrollment set, and outputs a vector containing candListLength objects of Candidates.
6.	// Destructor	
7.	};	

3.6.14.1. Video identification initialization

The function below will be called once prior to one or more calls of the searching function of Table 51. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 50 – VideoSearch::initialize

Prototype	ReturnCode initialize(const string &configDir, const string &enrollDir);	
		Input
		Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the VideoFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.6.15. Video identification search

The function below compares a proprietary identification template against the enrollment data and returns a candidate list.

Table 51 – VideoSearch::identifyVideo and VideoSearch::identifyImage

Prototype	ReturnCode identifyVideo(const PERSONREP &idVideoTemplate, const uint32_t candListLength, CANDIDATELIST &candList);		Searches a template generated from a ONEVIDEO against the enrollment set (video-to-video)
			Input
			Input
			Output
	ReturnCode identifyImage(const PERSONREP &idImageTemplate, const uint32_t candListLength, CANDIDATELIST &candList);		Searches a template generated from a MULTIFACE against the enrollment set (still-to-video)
			Input
			Input
			Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added		

	to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.	
Input Parameters	idTemplate	A template from generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.
	candListLength	The number of candidates the search should return
Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 3.6.7. Each candidate shall be populated by the implementation and added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
ReturnCode	Success	Success
	IdBadTempl	The input template was defective.
	Vendor	Vendor-defined failure

904 3.6.16. The ImageEnrollment Interface

905 The abstract class ImageEnrollment must be implemented by the SDK developer in a class named exactly
 906 SdkImageEnrollment.

	C++ code fragment	Remarks
1.	class ImageEnrollment	
2.	{	
	public:	
3.	virtual ReturnCode getPid(string &sdkId, string &email) = 0;	Return the sdk identifier and email
4.	virtual ReturnCode initialize(const string &configDir, const string &enrollDir, const uint32_t numPersons, const uint32_t numImages, const vector<string> &descriptions) = 0 ;	Initialize the enrollment session.
5.	virtual ReturnCode generateEnrollmentTemplate(const MULTIFACE &inputFaces, PERSONREP &outputTemplate) = 0;	This function takes a MULTIFACE (see 2.3.3) as input and outputs a proprietary template represented by a PERSONREP (see 3.6.5). For each input image in the MULTIFACE , the function shall return the estimated eye centers by setting PERSONREP .eyeCoordinates.
6.	// Destructor	
7.	};	

907 3.6.17. Implementation identifier

908 **Table 52 – ImageEnrollment::getPid**

Prototype	ReturnCode getPid(string &sdkId, string &email);	
		A developer-assigned ID. This shall be different for each submitted SDK.
		Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdkId	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email	Point of contact email address. The value cannot be the empty string.
ReturnCode	Success	Success

	Vendor	Vendor-defined failure
--	--------	------------------------

3.6.17.1. Initialization of the image enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-to-still.

Table 53 – ImageEnrollment::initialize

Prototype	ReturnCode initialize(const string &configDir, const string &enrollDir, const uint32_t numPersons, const uint32_t numImages, const std::vector<string> &descriptions);		
			Input
			Input
			Input
			Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called $N=1$ times by the NIST application immediately before any $M \geq 1$ calls to generateEnrollmentTemplate. The SDK should tolerate execution of $P > 1$ processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.		
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.	
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.	
	numPersons	The number of persons who will be enrolled.	
	numImages	The total number of images that will be enrolled, summed over all identities.	
	descriptions	A lexicon of labels one of which will be assigned to each enrollment image. EXAMPLE: The descriptions could be {"mugshot", "visa"}. NOTE: The identification search images may or may not be labeled. An identification image may carry a label not in this set of labels. The number of items stored in the vector is accessible via the vector::size() function.	
Output Parameters	none		
ReturnCode	Success	Success	
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.	
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).	
	InitNumData	The SDK cannot support the number of videos.	
	InitBadDesc	The descriptions are unexpected, or unusable.	
	Vendor	Vendor-defined failure	

3.6.17.2. Image enrollment

A **MULTIFACE** (see Table 13) is converted to a single enrollment template using the function below.

Table 54 – ImageEnrollment::generateEnrollmentTemplate

Prototypes	ReturnCode generateEnrollmentTemplate(const MULTIFACE &inputFaces, PERSONREP &outputTemplate);		
			Input
			Output
Description	<p>This function takes a MULTIFACE, and outputs a proprietary template in the form of a PERSONREP object. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function.</p> <p>If the function gives a non-zero exit status:</p> <ul style="list-style-type: none"> – If the exit status is ReturnCode::FailParse, NIST will debug, otherwise 		

	<ul style="list-style-type: none"> the test driver will ignore the output template (the template may have any size including zero) the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. <p>IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.</p>	
Input Parameters	inputFaces	An instance of a Table 13 structure.
Output Parameters	outputTemplate	An instance of a section 3.6.5 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the MULTIFACE . The eye coordinates shall be captured in the PERSONREP .eyeCoordinates variable, which is a vector of EYEPAIR objects. In the event the eye centers cannot be calculated, the SDK shall store an EYEPAIR and set EYEPAIR.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the MULTIFACE .
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of ONEVIDEO
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

916 3.6.18. The ImageFinalize Interface

917 The abstract class ImageFinalize must be implemented by the SDK developer in a class named exactly SdkImageFinalize.
 918 The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its
 919 manifest have been stored. These are described in section 2.4. The enrollment directory permissions will be read +
 920 write.

	C++ code fragment	Remarks
1.	class ImageFinalize	
2.	{	
	public:	
3.	virtual ReturnCode finalize(const string &enrollDir, const string &edbName, const string &edbManifest) = 0;	This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.
4.	// Destructor	
5.	};	

921 3.6.19. Finalize image enrollment

922 After all templates have been created, the function of Table 55 will be called. This freezes the enrollment data. After this
 923 call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search
 924 phases.

925 The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and
 926 data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data.
 927 No output is expected from this function, except a return code.

928 **Table 55 – ImageFinalize::finalize**

Prototypes	ReturnCode finalize(const string &enrollDir, const string &edbName, const string &edbManifest);	
		Input
		Input
		Input

Description	<p>This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 2.4. The enrollment directory permissions will be read + write.</p> <p>The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.</p> <p>This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.</p>	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 2.4. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 2.4. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

929 3.6.20. The ImageFeatureExtraction Interface

930 The abstract class ImageFeatureExtraction must be implemented by the SDK developer in a class named exactly
 931 SdkImageFeatureExtraction.

	C++ code fragment	Remarks
1.	class ImageFeatureExtraction	
2.	{	
	public:	
3.	virtual ReturnCode initialize(const string &configDir, const string &enrollDir) = 0;	Initialize the feature extraction session.
4.	virtual ReturnCode generateIdTemplate(const MULTIFACE &inputFaces, PERSONREP &outputTemplate) = 0;	<p>This function takes a MULTIFACE (see 2.3.3) as input and outputs a proprietary template represented by a PERSONREP (see 3.6.5).</p> <p>For each input image in the MULTIFACE, the function shall return the estimated eye centers by setting PERSONREP.eyeCoordinates.</p>
5.	// Destructor	
6.	};	

932 3.6.20.1. Image feature extraction initialization

933 Before one or more **MULTIFACES** are sent to the identification feature extraction function, the test harness will call the
 934 initialization function below.

935 **Table 56 – ImageFeatureExtraction::initialize**

Prototype	ReturnCode initialize(const string &configDir, const string &enrollDir);	
		Input
		Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by	

	the NIST application immediately before $M \geq 1$ calls generateIdTemplate. The SDK should tolerate execution of $P \geq 1$ processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

936 3.6.20.2. Image feature extraction

937 A **MULTIFACE** is converted to one identification template using the function below. The result may be stored by NIST, or
 938 used immediately. The SDK shall not attempt to store any data.

939 **Table 57 – ImageFeatureExtraction::generateIdTemplate**

Prototypes	ReturnCode generateIdTemplate(const MULTIFACE &inputFaces, PERSONREP &outputTemplate);	
		Input
		Output
Description	<p>This function takes a MULTIFACE (see 2.3.3) as input and populates a PERSONREP (see 3.6.5) with a proprietary template and eye coordinates.</p> <p>If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations.</p> <p>The function shall not have access to the enrollment data, nor shall it attempt access.</p>	
Input Parameters	inputFaces	An instance of a Table 13 structure.
Output Parameters	outputTemplate	An instance of a section 3.6.5 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the MULTIFACE . The eye coordinates shall be captured in the PERSONREP.eyeCoordinates variable, which is a vector of EYEPAIR objects. In the event the eye centers cannot be calculated, the SDK shall store an EYEPAIR and set EYEPAIR.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the MULTIFACE .
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of ONEVIDEO
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

940 3.6.21. The ImageSearch Interface

941 The abstract class ImageSearch must be implemented by the SDK developer in a class named exactly SdkImageSearch.

	C++ code fragment	Remarks
1.	class VideoFeatureExtraction	

2.	{	
3.	public:	
	virtual ReturnCode initialize(const string &configDir, const string &enrollDir) = 0;	Initialize the search session.
4.	virtual ReturnCode identifyVideo(const PERSONREP &idTemplate, const uint32_t candListLength, CANDIDATELIST &candList) = 0;	For video-to-still identification This function searches a template generated from an ONEVIDEO against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added to candList. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
5.	// Destructor	
6.	};	

942 3.6.21.1. Image identification initialization

943 The function below will be called once prior to one or more calls of the searching function of Table 59. The function might
944 set static internal variables so that the enrollment database is available to the subsequent identification searches.

945 **Table 58 – ImageSearch::initialize**

Prototype	ReturnCode initialize(const string &configDir, const string &enrollDir);	
		Input
		Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the ImageFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

946 3.6.22. Image identification search

947 The function below performs a video-to-still identification and compares a proprietary identification template generated
948 from a video against the enrollment data and returns a candidate list.

949 **Table 59 – ImageSearch::identifyVideo**

Prototype	ReturnCode identifyVideo(const PERSONREP &idVideoTemplate, const uint32_t candListLength, CANDIDATELIST &candList);		Searches a template generated from a ONEVIDEO against the enrollment set (video-to-still)
			Input
			Input
			Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.		
Input Parameters	idTemplate	A template from VideoFeatureExtraction::generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.	
	candListLength	The number of candidates the search should return	

FRVT 2012

Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 3.6.7. Each candidate shall be populated by the implementation and added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
ReturnCode	Success	Success
	IdBadTempl	The input template was defective.
	Vendor	Vendor-defined failure

950

951 **4. References**

FRVT 2002	Face Recognition Vendor Test 2002: Evaluation Report, NIST Interagency Report 6965, P. Jonathon Phillips, Patrick Grother, Ross J. Micheals, Duane M. Blackburn, Elham Tabassi, Mike Bone
FRVT 2002b	Face Recognition Vendor Test 2002: Supplemental Report, NIST Interagency Report 7083, Patrick Grother
FRVT 2006	P. Jonathon Phillips, W. Todd Scruggs, Alice J. O'Toole, Patrick J. Flynn, Kevin W. Bowyer, Cathy L. Schott, and Matthew Sharpe. "FRVT 2006 and ICE 2006 Large-Scale Results." NISTIR 7408, March 2007.
AN27	NIST Special Publication 500-271: American National Standard for Information Systems — Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information — Part 1. (ANSI/NIST ITL 1-2007). Approved April 20, 2007.
IREX III	P. Grother, G.W. Quinn, J. Matey, M. Ngan, W. Salamon, G. Fiumara, C. Watson, Iris Exchange III, Performance of Iris Identification Algorithms, NIST Interagency Report 7836, Released April 9, 2012. http://iris.nist.gov/irex
MBE	P. Grother, G .W. Quinn, and P. J. Phillips, Multiple-Biometric Evaluation (MBE) 2010, Report on the Evaluation of 2D Still Image Face Recognition Algorithms, NIST Interagency Report 7709, Released June 22, 2010. Revised August 23, 2010. http://face.nist.gov/mbe
MINEX	P. Grother et al., Performance and Interoperability of the INCITS 378 Template, NIST IR 7296 http://fingerprint.nist.gov/minex04/minex_report.pdf
MOC	P. Grother and W. Salamon, MINEX II - An Assessment of ISO/IEC 7816 Card-Based Match-on-Card Capabilities http://fingerprint.nist.gov/minex/minexII/NIST_MOC_ISO_CC_interop_test_plan_1102.pdf
PERFSTD INTEROP	ISO/IEC 19795-4 — Biometric Performance Testing and Reporting — Part 4: Interoperability Performance Testing. Posted as document 37N2370. The standard was published in 2007. It can be purchased from ANSI at http://webstore.ansi.org/ .
ISO STD05	ISO/IEC 19794-5:2005 — Information technology — Biometric data interchange formats — Part 5: Face image data. The standard was published in 2005, and can be purchased from ANSI at http://webstore.ansi.org/ Multipart standard of "Biometric data interchange formats". This standard was published in 2005. It was amended twice to include guidance to photographers, and then to include 3D information. Two corrigenda were published. All these changes and new material is currently being incorporated in revision of the standard. Publication is likely in early 2011. The documentary history is as follows. ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 1: Published 2008-07-01 International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 2: Published 2008-07-01 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 1: Conditions for taking photographs for face image data. Published 2007-12-15 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 2: Three dimensional image data. JTC 1/SC37/N3303. FCD text of the second edition. Contact pgrother AT nist DOT gov for more information.

952

Annex A

Submission of Implementations to the FRVT 2012

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: <http://www.nist.gov/itl/iad/ig/encrypt.cfm>

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

A.2 How to participate

Those wishing to participate in FRVT 2012 testing must do all of the following, on the schedule listed on Page 2.

- IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here. <http://www.nist.gov/itl/iad/ig/encrypt.cfm>
- Send a signed and fully completed copy of the *Application to Participate in the Face Recognition Vendor Test (FRVT) 2012*. This is available at <http://www.nist.gov/itl/iad/ig/frvt-2012.cfm>. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed FRVT 2012 Application to Participate shall be sent to NIST as a PDF.
- Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.
 - Encrypted data and SDKs below 20MB can be emailed to NIST at frvt2012@nist.gov
 - Encrypted data and SDKS above 20MB shall be

EITHER

- Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks, and then rename to include the filename extension need for passage through the NIST firewall.

```
you% split -a 3 -d -b 9000000 libFRVT2012_enron_A_02.tgz.gpg
```

```
you% ls -l x??? | xargs -iQ mv Q libFRVT2012_enron_A_02_Q.tgz.gpg
```

- Email each part in a separate email. Upon receipt NIST will

```
nist% cat frvt2012_enron_A02_*.tgz.gpg > libFRVT2012_enron_A_02.tgz.gpg
```

OR

- Made available as a file.zip.gpg or file.zip.asc download from a generic webserver¹⁷,

OR

- Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

FRVT 2012 Test Liaison (A203)
100 Bureau Drive
A203/Tech225/Stop 8940
NIST
Gaithersburg, MD 20899-8940
USA

In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296.

¹⁷ NIST will not register, or establish any kind of membership, on the provided website.

987 **A.3 Implementation validation**

988 Registered Participants will be provided with a small validation dataset and test program available on the website

989 <http://www.nist.gov/itl/iad/ig/frvt-2012.cfm> shortly after the final evaluation plan is released.

990 The validation test programs shall be compiled by the provider. The output of these programs shall be submitted to NIST.

991 Prior to submission of the SDK and validation data, the Participant must verify that their software executes on the
992 validation images, and produces correct similarity scores and templates.

993 Software submitted shall implement the FRVT 2012 API Specification as detailed in the body of this document.

994 Upon receipt of the SDK and validation output, NIST will attempt to reproduce the same output by executing the SDK on
995 the validation imagery, using a NIST computer. In the event of disagreement in the output, or other difficulties, the
996 Participant will be notified.